Erzeugende Gegnerische Netzwerke (GANs) für Inverses Design von RuO_2 Oberflächen

Generative Adversarial Networks (GANs) for Inverse Design of RuO_2 Surfaces

Wissenschaftliche Arbeit zur Erlangung des Grades

Master of Science

an der Fakultät für Chemie der Technischen Universität München.

Betreut von

Prof. Dr. Heiz

Lehrstuhl für Theoretische Chemie

Technische Universität München

Prof. Dr. Reuter

Theorie Abteilung am Fritz-Haber-Institut

Max-Planck-Gesellschaft

Dr. Christoph Scheurer

Gruppenleiter am Fritz-Haber-Institut

Max-Planck-Gesellschaft

Eingereicht von

Patricia König

12249 Berlin

Eingereicht am

Berlin, den 30.05.2022

Anhang I

Erklärung

Ich versichere hiermit, dass ich die von mir eingereichte Abschlussarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Berlin, 30.05.2022, Unterschrift

Abstract

Solving the longstanding puzzle of catalytically active RuO_2 structures in the oxidation of CO, has the potential to enable an efficient catalyst design for the conversion of CO exhausts in combustion processes. Reasons for the ongoing debate about catalytically active surface terminations of RuO_2 are that the experimental outcomes are strongly dependent on the catalyst pretreatment and the reaction conditions, as well as the limited computational resources in theoretical studies for the exploration of the RuO_2 potential energy surface. In recent theoretical studies, generative models have proven to be powerful tools for structure prediction of complex crystalline materials.

To tackle the vast chemical space of possible surface terminations, we present a Generative Adversarial Network (GAN) that is capable of cheaply generating diverse structural guesses for novel RuO_2 surface structures. Two training sets, one with 28,903 and the other with 18,944 RuO_2 surface terminations, were created with a grand-canonical basin hopping method and a Gaussian Approximated Potential, respectively. The atomic positions of these structures were mapped as Gaussian densities to a three-dimensional grid for the GAN input.

We demonstrate how two-dimensional images of RuO_2 structures with inferred lattice lengths and energy conditioning can be created in a two-dimensional Deep Convolutional Wasserstein-GAN (2D-DCWGAN) framework as a first step to realistic three-dimensional surface structures. The lattice lengths were predicted on-the-fly with two auxiliary networks in our GAN framework and the energy was ingrained in our latent space design. Additionally, the generation of realistic three-dimensional RuO_2 structures is incorporated in a three-dimensional Deep Convolutional Wasserstein-GAN (3D-DCWGAN) framework. These advances build the foundation which enables the implementation of realistic lattice lengths and an effective latent space design for the structure-energy-relationship in our 3D-DCWGAN framework in the future. Ultimately, these developments are necessary to produce reliable structural guesses for catalytically active surface terminations in the CO oxidation reaction.

Zusammenfassung

Das langwährende Rätsel der möglichen katalytisch aktiven RuO_2 Strukturen in der CO oxidation zu lösen, hat das Potential, ein effizientes Katalysatordesign für die Umwandlung von CO in Verbrennungsprozessen zu ermöglichen. Gründe für die fortwährende Diskussion um die katalytisch aktiven Oberflächenstrukturen von RuO_2 sind, dass die experimentellen Ergebnisse stark von der Katalysatorvorbehandlung und den Reaktionsbedingungen abhängen, sowie die limitierten Rechenkapazitäten in theoretischen Studien für die Erkundung der Potentialenergiehyperflächen von RuO_2. In kürzlichen theoretischen Studien, haben sich generative Modelle als leistungsfähige Werkzeuge für Strukturvorhersagen von kristallinen Materialien bewiesen. Wir haben deshalb ein gegnerisches generatives Netzwerk (Generative Adversarial Network) trainiert, das in der Lage ist, günstig vielfältige Strukturvorschläge für RuO_2 Oberflächenstrukturen zu generieren.

Um die volle Breite der möglichen Oberflächenstrukturen anzugehen, präsentieren wir ein gegnerisches generatives Netzwerk, das in der Lage ist, vielfältige Strukturvorschläge für RuO_2 Oberflächenstrukturen zu generieren. Zwei Trainingssets, eines mit 28.903 und das andere mit 18.944 RuO_2 Oberflächenstrukturen wurden jeweils mit einer großkanonischen Beckenhüpfenmethode (basin hopping) und einem Gauß approximierten Potential (Gaussian Approximated Potential) kreiert. Die Atompositionen dieser Strukturen wurden als Gaußdichten auf ein dreidimensionales Gitter für den Netzwerkinput abgebildet.

Wir zeigen, wie als erster Schritt zu realistischen dreidimensionalen Strukturen zweidimensionale Bilder von RuO_2 Strukturen mit erschlossenen Gitterlängen und Energiekonditionierung in einem zweidimensionalen tief gefalteten generierenden gegnerischen Wasserstein Netzwerk (2D-DCWGAN) erzeugt werden können. Die Gitterlängen wurden durch zwei Hilfsnetzwerke in unserem GAN Framework vorhergesagt und die Energy wurde die Gestaltung unseres latenten Raumes mitaufgenommen. Zusätzlich wurde die Generierung von realistischen dreidimensionalen RuO_2 Strukturen in unser dreidimensionales tief gefaltetes generierendes gegnerisches Wasserstein Netzwerk (3D-DCWGAN) eingebaut. Diese Fortschritte legen den Grundstein, der uns realistische Gitterlängen und eine effektive Gestaltung des latenten Raumes für die Struktur-Energie-Beziehung in unserem 3D-DCWGAN Framework ermöglicht. Desweiteren sind diese Entwicklungen notwendig, um verlässliche Strukturvorschläge für katalytisch aktive Oberflächenstrukturen in der CO Oxidationsreaktion erzeugen zu können.

Acknowledgements

First of all, thanks to Prof. Dr. Karsten Reuter for the opportunity to do my master thesis in his group. I greatly appreciated the opportunity to join the group seminar and therefore get an overview of the research done in the whole group, which was super insightful. Thanks to Dr. Christoph Scheurer for welcoming me in his subgroups and for the very helpful, constructive and fruitful discussions on my thesis and many thanks to Dr. Christoph Scheurer for not only being an extremely competent subgroup leader but also an understanding and empathic person. Thanks to Hanna Türk for being the best supervisor I could have asked for, patiently answering all my questions, giving me a lot of freedom in the meantime, and always having her office door open for me. Furthermore, I would like to thank Hanna Türk, Dr. Christoph Scheurer, Sina Stegmaier, Yonghyuk Lee and Simeon Beinlich for providing me with IT support and useful scripts during my thesis. A special thanks goes to Chiara Panosetti who created the basin hopping dataset for me which was the decisive factor for the convergence of my multi-GPU code. Many thanks to Hanna Türk, Konstantin Jakob, Yonghyuk Lee, Chiara Panosetti and Ryan Yang for reading my thesis, patiently correcting unnecessary errors, and their super helpful remarks! Finally, I want to express my gratitude for the endless support from my family and my friends.

Contents

List of Figures

List of Tables

List of Abbreviations

2D	Two-dimensional
3D	Three-dimensional
AI	Artificial intelligence
ADAM	Adaptive moment estimation optimizer
ASE	Atomic Simulation Environment
BCE	Binary cross entropy loss
CE	Cross entropy loss
CFD	Computational Fluid Dynamics
CNN	Convolutional Neural Network
CO	Carbon monoxide
DCGAN	Deep convolutional GAN
DDP	DistributedDataParallel Package
DFT	Density Functional Theory
DGM	Deep generative modeling
DIC	Delocalized internal coordinates
DP	DataParallel Package
EMD	Earth Mover's Distance
FGAN	Fisher GAN
GAP	Gaussian Approximation Potential
GAN	Generative Adversarial Network
GPR	Gaussian Process Regression
GPU	Graphics processing unit
INFOGAN	Information Maximizing GAN
JS	Jensen-Shannon divergence
KL	Kullback-Leibner divergence
LSGAN	Least squares GAN
ML	Machine learning
MLSM	MultiLabelSoftMargin Loss
MNIST	Modified National Institute of Standards and Technology database
MSE	Mean squared error
NN	Neural network
PES	Potential Energy Surface
RELU	Rectified linear units
SGD	Stochastic-gradient descent optimizer
SOAP	Smooth Overlap of Atomic Positions
TOF	Turnover frequency
WGAN	Wasserstein GAN
XRD	X-Ray Diffraction Spectroscopy
XRS	X-Ray Spectroscopy

1 Introduction

In most combustion processes of fossil fuels, carbon monoxide (CO), a toxic gas, forms as an unwanted by-product.[1] To reduce air pollution and avoid a threat to public health, gaseous exhausts must be converted to non-hazardous molecules before being released into the environment. As fossil fuels are still used as supports for the current energy supply, it is of utmost importance to design the disarming catalysts to act in the most efficient and environmentally-friendly manner as possible.[2] Two crucial factors for an efficient catalyst for the CO oxidation reaction are a high reactivity and selectivity towards CO.[3] In the past, three classes of CO oxidation catalysts have emerged. The first class consists of pure noble metals like Pt, Pd or Rh which fulfill the desired properties, but are on the other hand very expensive and limited in abundance.[3] As supported gold nanoparticles break and form bonds with oxygen easily,[4] they exhibit a high reactivity for the CO oxidation and form the second class of potential catalysts.[3] However, there are some challenges in the synthesis, pretreatment and activation of the nanoparticles to overcome.[4] The third class is represented by metal oxides.[3] Here, RuO_2 has proven as an efficient catalyst for the oxidation of CO[5–7]. Namely its low reaction barriers for the CO oxidation in oxidizing conditions make RuO_2 superior to Pt or Pd catalyst systems. However, due to the great number of relevant reaction parameters, the functionality of the RuO_2 catalyst is still illusive and its active surface conformation is still heavily debated.[5,8–17]

As an oxidic catalyst, the surface morphology of RuO_2 strongly depends on the reaction conditions, as well as its pretreatment.[5,8–17] This dependence led to two common problems in the early years of research on RuO_2: the materials gap and the pressure or temperature gap, as both, experimental and theoretical approaches, struggled to confidently introduce realistic pressure and temperature conditions to the catalytic system. On the experimental side, most analytic techniques were performed in ultra-high vacuum such as Scanning Electron Microscopy (SEM).[5] More recent approaches try to bridge the pressure gap by monitoring reaction characteristics with in-situ techniques, e.g. in-situ X-Ray Spectroscopy (XRS) or in-situ X-Ray Diffraction Spectra (XRD).[8,9] On the theoretical side, mainly density functional theory-based (DFT) structure computations were employed, which are usually performed at zero-temperature and zero-pressure. Reuter *et al.*[15–17] overcame this challenge by combining DFT with *ab initio* thermodynamics to efficiently compute equilibrium particle shapes of RuO_2 in both a pure-oxygen environment[15] and in a mixed oxygen and carbon monoxide environment.[16] The development of these methods that enable the access to the RuO_2 catalyst structures under operation conditions is of high importance as it links directly to the catalytic activity.

While the bulk structure of RuO_2 under reaction conditions has been identified and confirmed to be rutile early on,[17] heated discussions about the relevant surface terminations under realistic oxidation conditions continue.[7,8,15,18] Four crystal directions appear to be the most

relevant for catalysis and are thus discussed in the majority of studies.[5,7–18] Depending on the size of the unit cell and the slab thickness, each crystal direction can exist in different degrees of oxygen saturation which leads to hundreds of structure possibilities. In [18], Timmermann *et al.* performed a large-scale search for RuO_2 surface structures and discovered a variety of stable and novel surface terminations for RuO_2 in an O-poor environment. This study shows the potential of applying a suitable machine-learned, data-based approach to solve the question about the nature of the RuO_2 catalyst under reaction conditions. Timmermann *et al.* trained a Gaussian Approximated Potential (GAP) to build a surrogate model for the potential energy surface (PES) on the subspace of $p(1 \times 1)$ surface structures.[18] Due to the restricted structural possibilities of the investigated (1×1) surface unit cells, we believe that more metastable and novel stable surface terminations can be found using larger structural motifs to elucidate on the reaction mechanism.

In recent studies, Generative adversarial networks (**GAN**s) have successfully been used for the exploration of chemical spaces and for the generation of novel crystalline materials with tailored properties.[19,20] For example in a theoretical study, Kim *et al.* created a GAN framework to predict 121 novel zeolite structures with distinct methane heat absorption values.[20] In general, GANs are a data-driven deep learning approach[19] that can speed up large-scale material development by identifying patterns in a database of already confirmed structures.[21] On top of generating new compositions or polymorphs, the output of a GAN can be tuned by additional input data, e.g. energetical information of structures, which makes user-desired properties in the generation of data accessible.[19,20]

In this thesis, a data-driven approach is developed to unravel the catalytically active structures of the CO oxidation on RuO_2 by training a GAN and to facilitate the search for more efficient catalysts in the future. For the training process, two different databases are created - one based on the GAP structures from Timmermann *et al.* in [18] and one based on the grand-canonical basin hopping method in Delocalized Internal Coordinates, as developed by Panosetti *et al.* in [22, 23] - and are used to explore the potential of generative adversarial networks (GANs) by learning the underlying probability distribution of the RuO_2 data space. Ideally, after a converged training process the output of the generative model can be converted to Cartesian coordinates and geometry optimized to verify whether a new basin on the PES of RuO_2 is reached. In the context of the structural surface exploration on RuO_2, the GAN can thus serve as an effective supplier of new structural guesses based on the data input. With tailored dataset design and a subsequent structure optimization, we aim to use a GAN as a powerful tool to bridge the materials gap in theoretical studies.

2 Theoretical Framework

RuO_2 catalysts have proven to be efficient in the oxidation reaction of CO and methanol on RuO_2.[5–7,24,25] However, the surface morphology of the active catalyst strongly depends on the environmental conditions during the reaction as well as its pretreatment and is subject to an ongoing debate.[5,8] In the following, an overview of existing research on the RuO_2 catalyst system and its surface morphology will be given. The aim of this thesis is to develop a GAN that contains the features of a RuO_2 dataspace, for which basic and more advanced concepts of neural networks will be explained afterwards.

2.1 RuO_2

The exploration of the crystal structure of RuO_2 is based on an interplay of experimental and theoretical approaches.[5,8–18] In the following, RuO_2 in the form of rutile is introduced and different proposed, catalytically active surface terminations exposed to various atmospheres and reaction conditions are discussed in context of the CO oxidation.

2.1.1 Rutile

At ambient conditions, RuO_2 crystallizes in the rutile structure. Fig. 2.1 illustrates a unit cell of a rutile crystal, in which each Ru atoms has six O neighbors and each O atoms has three Ru neighbors. The O atoms form a slightly distorted octahedron with four basal (bond length $2.00\,\text{Å}$) and two apical (bond length $1.96\,\text{Å}$) O atoms.[17]

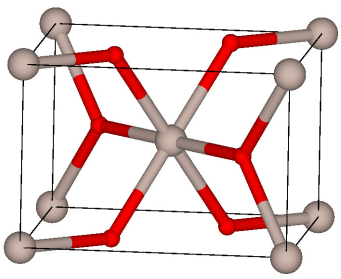


Figure 2.1: Unit cell of rutile RuO_2. Ru atoms are drawn as grey spheres and O atoms as red spheres.

Different surface orientations can be created by cutting a crystal along a certain direction, defined by lattice planes. These lattice planes are characterized by the Miller indices (hkl) and denote where the lattice vectors $\vec{a_1}$, $\vec{a_2}$ and $\vec{a_3}$ of the cell are cut at positions $\frac{1}{h}\vec{a_1}$, $\frac{1}{k}\vec{a_2}$ and $\frac{1}{l}\vec{a_3}$.

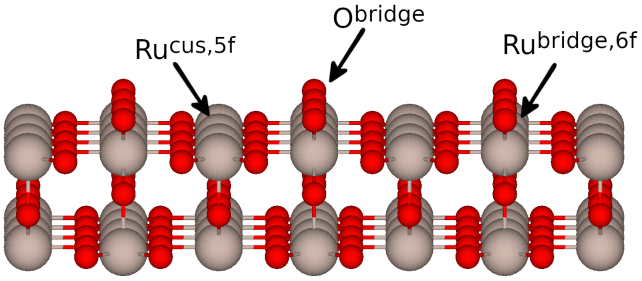
A set of symmetry equivalent lattice planes is denoted as $\{hkl\}$.[26] Additional atoms on top of the regular monomeric unit of a surface structure can form an superimposed structure termed as a superstructure. In case, the superstructure forms a primitive rectangle of cell dimensions $2\vec{a_1}$, $2\vec{a_2}$, the superstructure of the (hkl) lattice plane is denoted as a $p(2 \times 2)(hkl)$. In case, this superstructure is aligned as a rectangle with an additional atom in the center, it is termed as $c(2 \times 2)(hkl)$. For a detailed instruction on Wood's nomenclature for surface superstructures, the reader is referred to [27].

One systematic approach to obtain the subspace of $p(1 \times 1)RuO_2$–cells is to cut the rutile crystal along the (001), (100), (101), (110) and (111) directions.[15] For all of these five surface orientations, O-poor, stoichiometric and O-rich terminations can be found.[18] These surface terminations have been identified to be the most relevant for catalysis.[5,7–18] One exception is the (111) surface, where an additional super-O-rich termination was found to be catalytically relevant.[15] In general, for any surface structure, additional super-O-rich or super-O-poor termination can be build that contain more or less O atoms than the O-rich or O-poor structure, respectively which gives rise to hundreds of possible surface configurations. In theoretical studies, the structures are normally limited to $p(1 \times 1)$–cells or small (2×1)–supercells.[18]

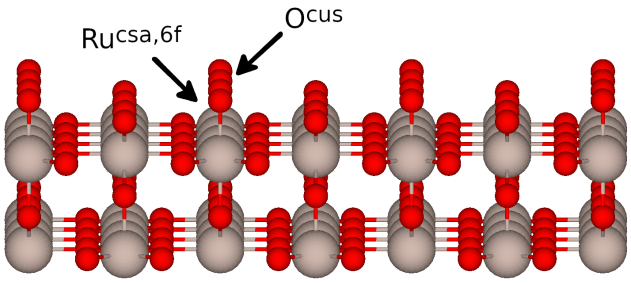
Fig. 2.2 illustrates three exemplary cuts along the (110) direction of rutile, resulting in an O-poor, a stoichiometric and an O-rich termination. After every cut in (110) direction, the repeating structural motif of O – RuO – O trilayers is found.[17] Consequently, depending on the truncation of the atom type, three different surface terminations can be obtained for $RuO_2(110)$.[17]

$RuO_2(110)$–O^{bridge} (see Subfig. 2.2a) is a stoichiometric termination where the ratio between terminating Ru and O atoms is 1:2, like the bulk ratio of RuO_2. The sixfold coordinated Ru atoms are saturated in terms of their coordination number whereas the fivefold coordinated Ru atoms lack one O atom on top and are thus termed coordinatively unsaturated (cus) sites. The coordinatively saturated sites are termed as bridge sites because the O-adsorbate at this site is bridging two adjacent Ru atoms. This termination contains no charge and is thus also considered as the most stable termination.[17] In contrast to $RuO_2(110)$–O^{bridge}, $RuO_2(110)$–O^{cus} (see Subfig. 2.2b) forms a polar surface.

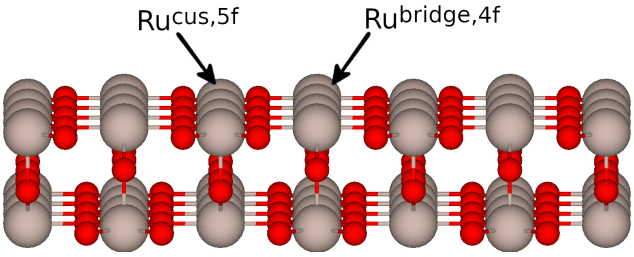
Here, all Ru atoms are coordinatively saturated (cas). The onefold coordinated O atoms on top of the formerly $Ru^{cus,6f}$ sites (now $Ru^{cas,6f}$) are termed as O^{cus} sites. Due to the excess of oxygen atoms, this surface termination is referred to as O-rich surface structure. The last termination - $RuO_2(110)$-Ru - consists of a RuO plane with only four- and fivefold coordinated Ru atoms and forms a polar surface as well. Remaining O atoms lie in the plane and are threefold coordinated.[17] Since in this termination the number of Ru atoms larger than twice the number of O atoms, it is termed as metal-rich or O-poor surface.



a) $RuO_2(110)$–O^{bridge} surface termination with alternating fivefold and sixfold coordinated Ru atoms on cus and bridge sites, respectively. Onefold coordinated O atoms are located on top of the sixfold coordinated Ru atoms forming bridge sites.[17]



b) $RuO_2(110)$–O^{cus} with additional onefold coordinated O atoms. The onefold coordinated O atoms on top of the former $Ru^{cus,6f}$ sites (now $Ru^{cas,6f}$) are termed as O^{cus} sites. All Ru atoms are sixfold coordinated and thus coordinatively saturated.[17]



c) $RuO_2(110)$–Ru termination with O atoms and four- and fivefold coordinated Ru atoms.

Figure 2.2: Different surface terminations of rutile (110). Ru atoms are drawn as grey spheres and O atoms as red spheres. Structures recreated from [17].

2.1.2 Catalytically active species for the CO oxidation reaction

In the following, we will focus on the main historical steps of unravelling the reaction mechanism of the CO oxidation ($RuO_2 + 2\,CO \longrightarrow Ru + 2\,CO_2$). An overview of the timeline of the following studies is provided in Fig. 2.3. Back in 1975, it was falsely assumed that the active species during the CO oxidation is the Ru(0001) surface.[28] However, it was soon discovered that an ultra thin RuO_2 layer forms during the reaction and is responsible for the catalytic activity.[5] This misinterpretation of data in [28] originated as chemisorbed oxygen was not accessible in

ultra-high vaccuum conditions, which is now known as the pressure-gap in experiments.[8] In 1986, (1x1) chemisorbed oxygen on Ru was first proposed.[12] About ten years later, a different study[10,14] found that subsurface oxygen dissolved in ruthenium and surface oxides drive the reaction. In 2005, another model was proposed in [13] where a 1 nm thick RuO_2 layer covering a metallic Ru core is assumed to be the catalytic species. Therein, the RuO_2(110) and (100) facets form as layers under oxygen exposure and render undercoordinated Ru atoms responsible for the catalytic activity.[11] In 2009 and 2011, two subsequent studies investigated the influence of different CO- and O-gas ratios during the catalyst preparation and, concomitantly, RuO_2(101) and (111) were identified as additional possible active facets.[8,9]

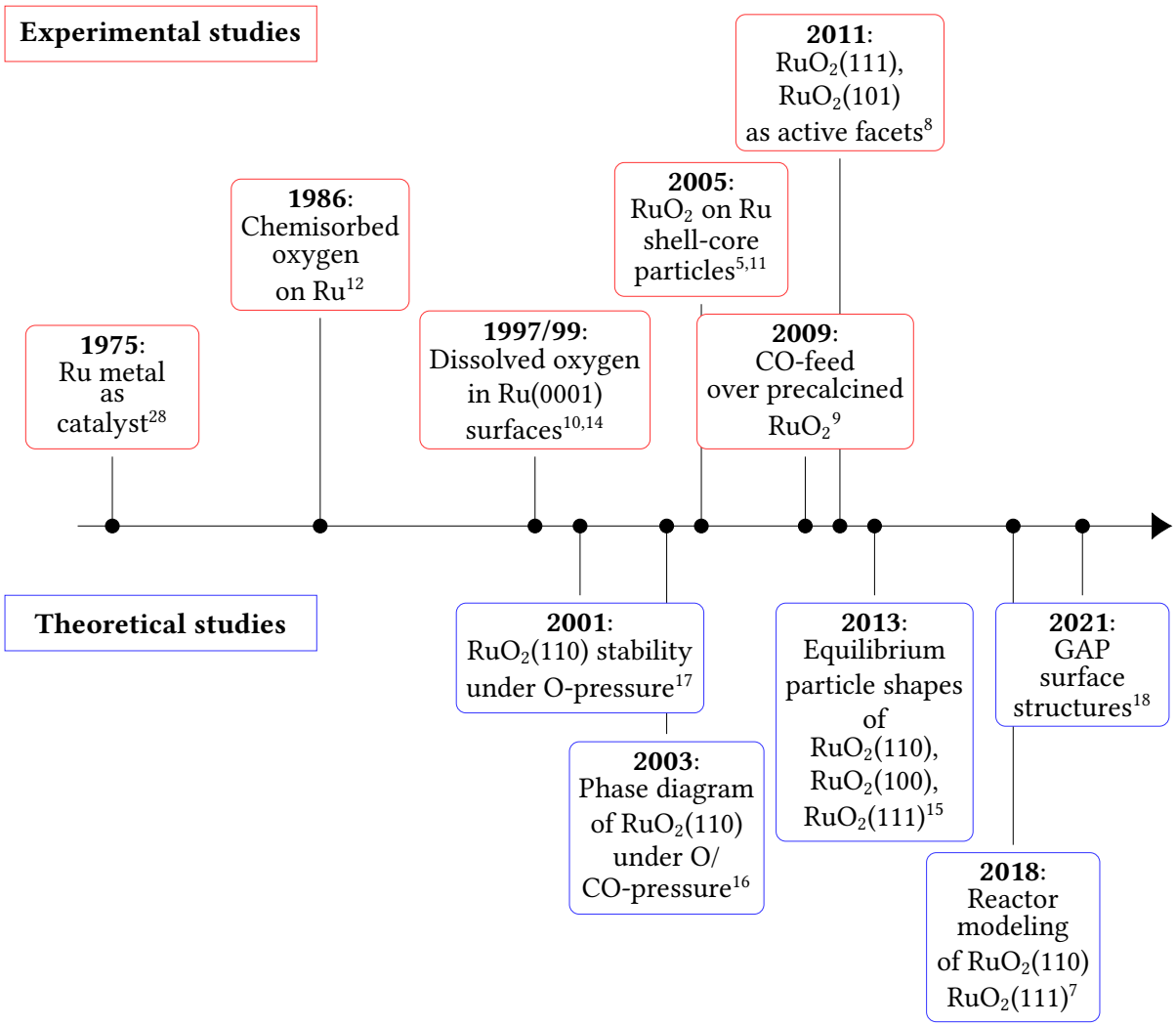


Figure 2.3: Timeline of the most relevant discoveries of RuO_2 structures for the CO oxidation.

Adjusting theoretical models to the experimental reality poses a challenge for theorists.[15–18] In 2001, Reuter *et al.* first computed the stability of the three surface terminations of RuO_2(110) (see Fig. 2.2) under variation of oxygen pressure.[17] In a follow-up study, they calculated the phase diagram for the same RuO_2(110) model in an O and CO environment and proposed possible reaction pathways.[16] Computing first-principles atomistic thermodynamic surface free

energy constructions of RuO_2 in oxygen environments, Reuter *et al.* explored the pretreatment-morphology relationships and discovered a strong structure dependence of the CO oxidation activity.[15] In 2018, a first-principles kinetic Monte Carlo approach was coupled with Computational Fluid Dynamics (CFD) to study the reactivity of RuO_2(110) and (111) facets in a reactor.[7] Contrary to previous models, Timmermann *et al.* developed a data-driven approach towards surface exploration of RuO_2 and discovered a variety of novel surface terminations in an O-poor environment.[18]

Based on these theoretical and experimental studies, we will discuss possible catalytically active structures of RuO_2 in the following. As shown in Fig. 2.4, not all three surface terminations of all five possible lattice planes were found to be of importance for the CO oxidation reaction.

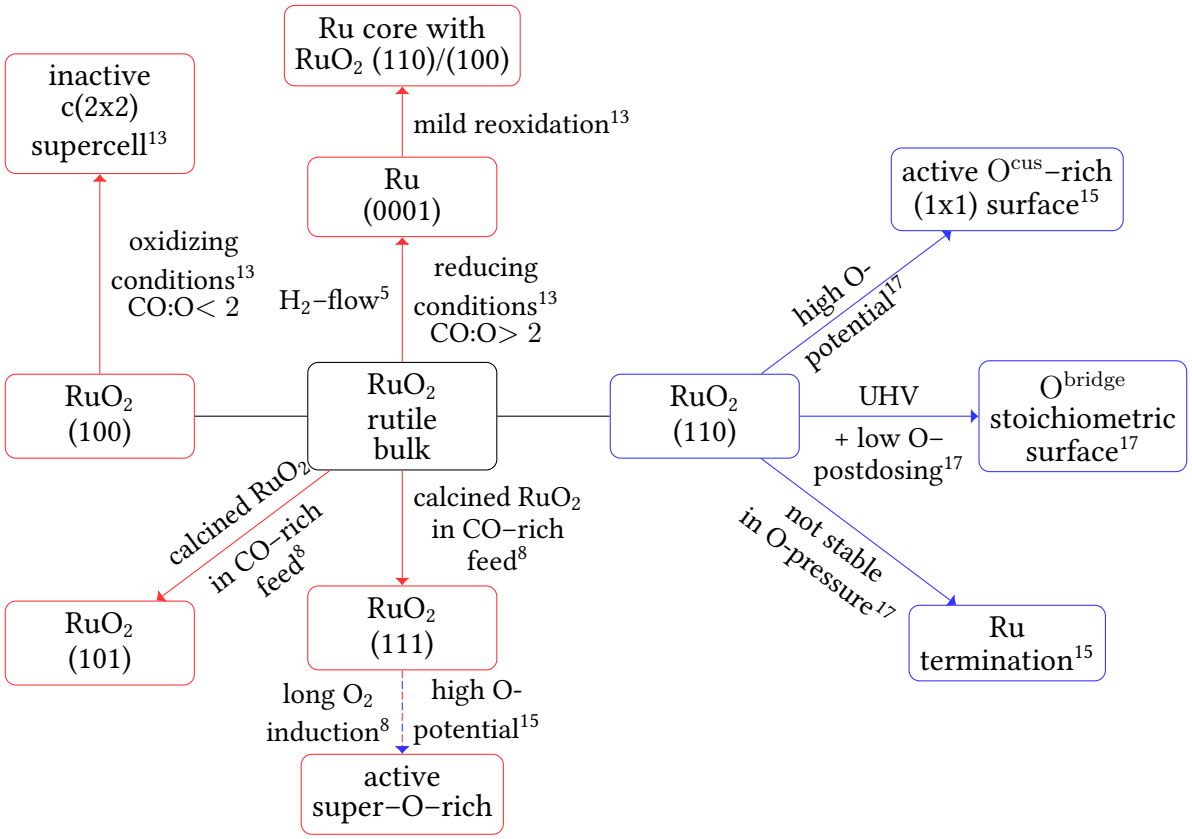


Figure 2.4: Overview of the most relevant surface terminations and structures of RuO_2 and transitions in-between. Results from theoretical studies are highlighted in blue and experimental outcomes in red.[8,13,15,17]

In several experiments, the RuO_2(110) and (100) were found to be reactive surface terminations that form under oxygen exposure and render undercoordinated Ru atoms responsible for the catalytic activity.[5,8,11,13] In [13], bulk RuO_2 is converted to Ru(0001) under reducing conditions and then reoxidized under mild conditions ($T < 500K$). Upon this pretreatment, thin RuO_2(110) and (100) layers form around Ru particles. In theoretical studies, the formation of RuO_2 facets is investigated with respect to their thermodynamic stability and proposed as an al-

ternative to the experimentally introduced core-shell model.[15–17] In [17], the $RuO_2(110)$–O^{bridge} surface termination from Subfig. 2.2a was found to be stable in UHV whereas $RuO_2(110)$–O^{cus} in Subfig. 2.2b becomes stable under high oxygen pressure.[17] Upon formation of $RuO_2(110)$–O^{cus}, up to 10% vacancies can be introduced,[17] which poses the question of further existing catalytic structures that might not have been discovered yet. As already written above, the $RuO_2(110)$–O^{cus} surface termination has an inherent dipole moment due to the O^{cus} additional atoms. This leads to a relaxation of bonds in proximity to the surface and thus to properties different from the bulk structure.[17] In the same study, they also introduced an O-poor limit for the minimal oxygen pressure at which metallic Ru would be formed and an O-rich pressure limit at which oxygen would start to condense on the surface.[17] Both configurations, $RuO_2(110)$–O^{bridge} and $RuO_2(110)$-O^{cus}, are considered active. For the $RuO_2(110)$–O^{cus}, a higher catalytic activity is expected due to single-coordinated oxygen atoms on the surface. In this *ab-initio* thermodynamic approach, the $RuO_2(110)$–Ru is not predicted as a stable configuration in oxygen environment. Later however, in the subsequent study in 2003, they introduced a multi-gas environment for the investigation of a $RuO_2(110)$ surface with respect to O_2 and CO pressures in which they also computed the phase diagram for prominent absorption structures and discussed possible reaction mechanisms. They predict high turnover numbers at gas conditions that enable regions of phase coexistence where the reactions of $CO^{cus} + O^{cus} \rightarrow CO_2$ and $CO^{cus} + O^{bridge} \rightarrow CO_2$ are assumed to be the most active. Furthermore, they estimated a limit for the CO pressure, at which RuO_2 would be reduced to Ru.[16]

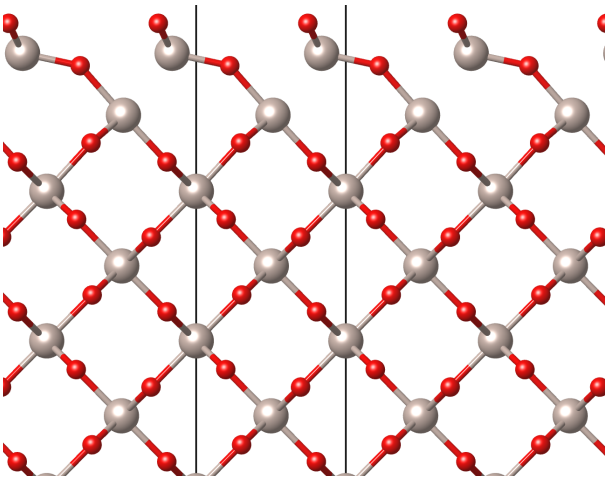


Figure 2.5: Novel, stable surface structure of $RuO_2(010)$ for the oxygen-poor termination as discovered by Timmermann *et al.* in [18].

In experiments, catalyst pretreatment has a significant influence on the catalytic reactivity and thus the turnover frequencies (TOFs). A preparation in O_2-rich atmosphere led to an initially inactive RuO_2 catalyst towards CO oxidation whereas a preparation in a CO-rich atmosphere resulted in $RuO_2(111)$ and $RuO_2(101)$ as active catalytic species.[8] These two structures were coined as potentially novel, catalytically active surfaces. The study assumes that upon a longer oxygen induction period on $RuO_2(111)$, the super–O–rich termination is formed. This is in good agreement with the computed equilibrium particle shape for high oxygen chemi-

cal potentials on RuO_2(111).[15] The formation of RuO_2(101), however, is contradictory to the thermodynamically expected particle shape. Reuter *et al.* interpreted its discovery as a kinetic artefact due to the experimental conditions.[15] In general, different pretreatments - such as different calcination temperatures of RuO_2 - resulted in different catalytic activities measured in a variety of experiments.[5,8,9] It is thus important to understand the involved facets and structure systems to tune the catalyst pretreatment with respect to the most efficient catalytic outcome. A further example why the structural knowledge of a catalyst is important is the formation of the c(2x2)RuO_2(100) facet. This facet was found to be inactive towards oxygen dissociation, thus inhibiting the CO oxidation once formed.[5,15] In experiments, it was recorded to form under oxidizing conditions of the RuO_2(100) facet. Further, the presence of moisture is believed to inhibit the dissipative adsorption of oxygen because coordinatively unsaturated Ru sites were blocked by water molecules.[5] In this spirit, the relationship between the structural and catalytic characteristics can not only be used to improve a catalyst's efficiency but also to prevent deactivation of the catalyst.

The latest approach from Timmermann *et al.* used a GAP to approximate the PES of RuO_2 by a data-efficient protocol which led to a variety of novel and metastable surface terminations in an O-poor environment.[18] One of those structures is illustrated in Fig. 2.5. As summarized above, there is a plethora of presumed active structures and this list might not even cover the full chemical space of involved structural possibilities yet. Thus, this study aims to train a generative model to explore the PES in a more thorough manner.

2.2 Neural Networks

In this section, the basics of neural networks in particular with convolutional layers will be introduced to then explain the concept of generative adversarial networks and specifically the Wasserstein-GAN as an improvement of *vanilla* GANs.

Historically, the term *machine learning* was first used by Arthur Samuel to describe the learning process of a program that produces unexpected and beneficial outcomes to solve a problem it was not explicitly designed for in the first place.[29] During this process, the program learns to extract basic patterns from a **training data** set. The prediction error between the program's output and a **test data** set is quantified with a metric or **cost function**.[30] This prediction error then helps the program to improve its performance by adjusting internal parameters accordingly.[29]

One main goal of artificial intelligence (**AI**) is to classify and cluster data. AI explores hidden patterns within data sets by use of different machine learning algorithms, e.g. multivariable regression.[30] Neural Networks (**NN**s) constitute a group of machine learning algorithms mimicking the cerebral structures of animals with connected processors that are referred to as **neurons**. Once successfully trained, neural networks present powerful tools applicable to a multitude of optimization and prediction problems across all scientific fields. For example, a very recent study published in 2020 utilizes a neural network-based technique to solve the electronic Schrödinger equation efficiently by predicting the related wave function.[31] One exemplary structure of a neural network is demonstrated in Fig. 2.6. Colored spheres represent single neurons that are vertically aligned in layers. The first layer is called the **input layer** because it is fed the data set. Here, each input entry x_i is connected with all other entries (or neurons $h_i^{(1)}$) in the next layer.

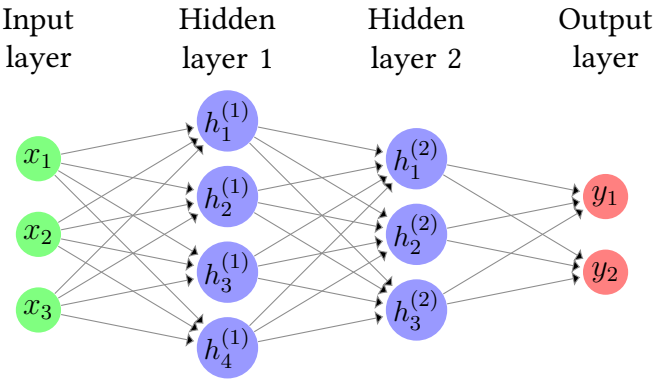


Figure 2.6: An example structure for a fully-connected multilayer perceptron with one input layer (green), two hidden layers (blue) and one output layer (red). The grey arrows show the connectivity of the neurons and weights tune the significance for each entry in the network.

The significance of neuron $h_i^{(k)}$ is tuned by the **weight** $w_{i,j}^{(k)}$ that connects the neuron $h_i^{(k)}$ with neuron $h_j^{(k+1)}$. Connectivities between neurons are indicated as grey arrows in Fig. 2.6. In neural network layers, neurons can be fully connected, meaning that each neuron receives

information from all neurons in the previous layer and passes the computed information to all neurons in the next layer.[32] The last layer is termed as **output layer** as it consists of the desired output data of the neural network. All layers between input and output layer are referred to as **hidden layers** since their entries are normally not accessed by the user. The amount of neurons and layers that leads to the most accurate results is problem-specific and has to be adjusted during the learning process. A neural network with three different layer types is called a **multilayer perceptron**.[32,33]

As for all machine learning algorithms, the parameters of a neural network are not fine-tuned to yield the desired output right from the start. Rather, the performance of the neural network is optimized during the training process. Input data is passed through the network in the so-called forward pass in order to compute the output data. The prediction error of the network output is then quantified with a cost function. In the backward pass, an optimizer adjusts the weights to achieve better results. In the following subsections, the steps of the training process are explained in detail.

2.2.1 Forward Propagation

The process of feeding data to the neural network and computing the output is termed **forward propagation**. During a training process, the training set is divided into smaller parts, so-called **batches**, to update the NN regularly. One cycle over the whole training set is referred to as an **epoch**.[32,33]

In the following, the forward propagation process through a fully-connected neural network will be exemplified for the network shown in Fig. 2.6. All entries of the hidden layer $\vec{h}^{(1)}$ are computed as a matrix-vector-multiplication of the input vector $\vec{x}$ with the weight matrix $\mathbf{W}$:

$$\vec{h}^{(1)} = \mathbf{W} \cdot \vec{x} \quad . \tag{2.1}$$

In this two-dimensional example, the dimensions of the first weight matrix are restricted to the size of the first hidden layer $\times$ size of input layer. This consecutive scheme of feeding the output of the previous layer as an input to the next layer and computing a new output with a new weight matrix is continued until the output layer is reached. During the training process of the neural network, the weights and an optional **bias** term b ($\vec{h}^{(1)} = \mathbf{W} \cdot \vec{x} + b$) are optimized until the error of network output is sufficiently small in comparison to the test data set. Weights indicate the influence of each input feature in predicting the final output, similar to a multivariate regression. Additionally, the bias term can be added to all entries after each vector-matrix multiplication to shift the outputs in analogy to an intercept in a linear regression.[32]

Activation Functions

During the forward propagation, a non-linear **activation function** $R(z)$ can be applied to the output of a layer before passing it on to the next layer. The main purpose of an activation function is to improve the learning behavior of **complex non-linear** patterns in a data set.

Activation functions map the output data of a layer, i.e. $W \cdot \vec{x} + b$, to a **fixed range** and thus facilitate the training of neural networks. As it will be explained in the section on the backward pass, the derivative of the activation functions must be defined at all points and activation functions must thus be **continuously differentiable**.[32,34]

Typical activation functions are **RELU** (rectified linear units) or **LeakyRELU**. LeakyRELU is defined as:[35]

$$R_{\text{Leaky}}(z) = \begin{cases} z & z > 0 \\ \alpha \cdot z & z \leq 0 \end{cases} \tag{2.2}$$

and its derivative as:

$$R'_{\text{Leaky}}(z) = \begin{cases} 1 & z > 0 \\ \alpha & z \leq 0 \end{cases} \quad . \tag{2.3}$$

An advantage of LeakyRELU over RELU is the small constant slope α ($0 < \alpha \ll 1$) which avoids that the derivatives become zero and allows for negative activations in the network as shown in Fig. 2.7a.[35,36] In RELU, this slope is chosen as $\alpha = 0$.

A more computationally costly activation function is the **Sigmoid** activation function:

$$S(z) = \frac{1}{1 + e^{-z}} \quad . \tag{2.4}$$

Its derivative reads:[32]

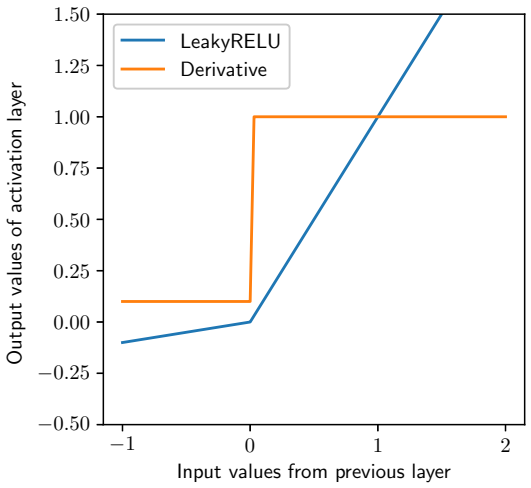
$$S'(z) = \frac{1}{1 + e^{-z}} \left(1 - \frac{1}{1 + e^{-z}}\right) = S(z) \cdot (1 - S(z)) \quad . \tag{2.5}$$

Both, the Sigmoid activation function and its derivative are shown in Fig. 2.7b.

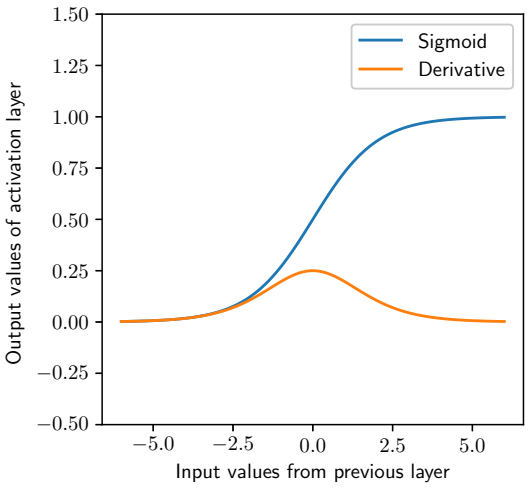
Normalization layers

One possibility to further stabilize the training process in neural networks is offered by the introduction of **normalization layers**.[37,38] Although there exists a variety of different forms, the two most prominent are **batch** and **layer normalization**. **Batch Normalization** accelerates the convergence of the network by reducing internal covariate shifts inside each batch B of size m,[37] whereas layer normalization normalizes K features within a single training case i.[38]

Both techniques compute the mean μ_B or μ_i and the variance σ_B^2 or σ_i^2, respectively. In batch normalization, each scalar entry in the batch $x_{B,\text{i}}$ is normalized to zero mean and standard deviation, whereas in layer normalization all K elements in the sample x_i have zero mean and unit variance. Both transformations are conducted in NNs by two learnable parameters



a) A plot of the LeakyRELU activation function and its derivative.



b) A plot of the Sigmoid activation function and its derivative.

Figure 2.7: Two activation functions and their derivatives.

$\gamma = \sqrt{\sigma_B^2}$ or $\gamma = \sqrt{\sigma_i^2}$ and $\beta = \mu_B$ or $\beta = \mu_i$, respectively. These parameters allow for reversibility because they can be 'forgotten' by the network if they are unfavorable for the training process. Table 2.1 displays the formulas used for the normalizations.[37,38]

Table 2.1: Batch and Layer Normalizing Transforms, applied to an activation x_i over a mini-batch B or over the features K, respectively.[37,38]

Batch Normalization	Layer Normalization
$\mu_B = \frac{1}{m}\sum_{i=1}^{m} x_{B,i}$	$\mu_i = \frac{1}{K}\sum_{k=1}^{K} x_{i,k}$
$\sigma_B^2 = \frac{1}{m}\sum_{i=1}^{m} (x_{B,i} - \mu_B)$	$\sigma_i^2 = \frac{1}{K}\sum_{k=1}^{K} (x_{i,k} - \mu_i)^2$
$\hat{x}_{B,i} = \frac{x_{B,i} - \mu_B}{\sqrt{\sigma_B^2}}$	$\hat{x}_{i,k} = \frac{x_{i,k} - \mu_i}{\sqrt{\sigma_i^2}}$
$y_{B,i} = \gamma \cdot \hat{x}_{B,i} + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_{B,i})$	$y_i = \gamma \cdot \hat{x}_i + \beta \equiv \mathrm{LN}_{\gamma,\beta}(x_i)$

Put in a nutshell, batch normalization performs a feature-by-feature normalization whereas in layer normalization the normalization is performed across all different features. As layer normalization is independent of the batch-sizes, it can lead to a more stable training behavior for small batch sizes.[37,38]

2.2.2 Backward Propagation

In **supervised learning**, the predicted output y of a neural network is compared to the expected output value $\hat{y}$ of the training data set. During training, the deviation between the expected value $\hat{y}$ and the predicted output y is quantified by a **cost** or **loss function** after each forward pass that accumulates the **loss** of a neural network resulting from the prediction. During the **backward pass**, however, the partial derivatives of each NN layer with respect to their weights are computed and grouped together as **layer errors**. These layer errors are then passed on to the previous layer until the input layer is reached. Ultimately, a **gradient-based optimization algorithm** uses the computed gradients and updates the weights in order to improve the NN's output for the next forward pass and concomitantly reduce the loss of the neural network.[32,39]

Loss Functions

The choice of an appropriate **loss** or **cost functions** depends on the type of optimization problem, such as regression or classification of data. In regression problems, a continuous quantity is predicted, e.g. the salary of a person or the price of a product, whereas in classification problems a discrete label is predicted for an input sample, e.g. if a person has health insurance or not or if an outcome is true or false.

A commonly used loss function for regression problems is the **mean squared error** (MSE):

$$\text{MSE} = \frac{1}{n}\sum_{\text{i}=1}^{n}(\hat{y} - y)^2 \quad , \tag{2.6}$$

where n is the size of the training set while y and $\hat{y}$ are the predicted and real value of the continuous output quantity, respectively.[32,40]

In a binary classification problem, samples are assigned one of two labels in the dataset. To quantify the prediction error of the discrete output in comparison to the expected labels, the **binary cross entropy loss** (BCE) or **log loss** function is prominently used:[39,41]

$$\text{BCE} = -(y \cdot \log(p)) + (1 - y) \cdot \log(1 - p) \quad . \tag{2.7}$$

In Eq. (2.7), the network output y is the discrete binary indicator with values being either 0 or 1, and p is the probability to observe the corresponding label in each epoch, i.e. $p(y = 0) = \frac{n_{y=0}}{n}$ or $p(y = 1) = \frac{n_{y=1}}{n}$, where $n_{y=0}$ and $n_{y=1}$ count the number of correct outcomes for $y = 0$ and $y = 1$, respectively. In Fig. 2.8, the BCE for the label $y = 1$ is plotted.

It is evident that the log loss of a perfect model ($p = 1, y = 1$) is zero. BCE thus penalizes if the predicted probability increases for the wrong label and if the predicted probability decreases for the correct label.[41]

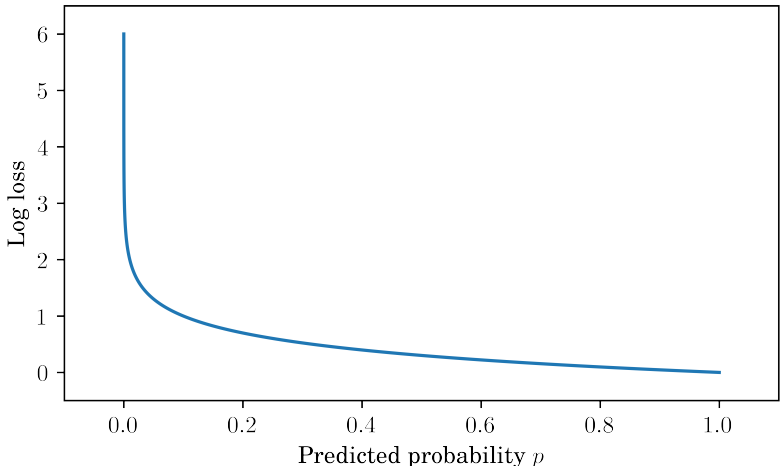


Figure 2.8: Plot of the binary cross entropy loss function if label $y = 1$.

In multi-class classification problems, samples can be assigned to one of $n > 2$ class labels. For such a problem, a more general **cross entropy loss** (CE) for one sample $y_{i,c}$ classified with the label c can be computed

$$\text{CE} = -\sum_{c=1}^{n} y_{i,c} \log(p_{i,c}). \quad . \tag{2.8}$$

CE averages the difference between actual and predicted probability distribution over all possible classes.[39]

In multi-label classification problems, each sample is assigned zero or more labels as a function of the input data. Here, a possible loss function is the *MultiLabelSoftMarginLoss* (MLSM):

$$\text{MLSM}(x, y) = -\frac{1}{C} \sum_{i} \left[y_i \cdot \log \left(\frac{1}{1 + \exp(-x_i)} \right) + (1 - y_i) \cdot \log \left(\frac{\exp(-x_i)}{1 + \exp(-x_i)} \right) \right] \quad , \tag{2.9}$$

where input x and target y are of size (N, C).[42] MLSM takes an average of the cross entropy loss (see Eq. (2.8)) across the multiple class labels. An additional sigmoid function (see Eq. (2.4)) is applied to improve the scale of this loss function because the loss increases exponentially for wrong classification.

Optimizers

As already described, the gradient of the cost function $\nabla_W C(W, x_i, y_i)$ at the current system parameters (weights and bias terms) is computed in the backpropagation. A gradient-based

optimization algorithm or short **optimizer** updates the weights and bias terms in each layer in order to optimize the NN's output. Here, the gradient of the cost function indicates the step direction and the **learning rate** λ indicates the step size in each optimization step.[43]

$$\mathbf{W}_{new} = \mathbf{W} - \lambda \cdot \nabla_{\mathbf{W}} C(\mathbf{W}, x_i, y_i) \quad . \tag{2.10}$$

This optimizer is called stochastic gradient descent (SGD) as samples and weights are drawn randomly from the whole training set.[43]

The Adaptive Moment Estimation (**Adam**) optimizer is an extension to SGD which can handle sparse gradients on noisy problems and computes adaptive learning rates for each parameter to improve the learning process.[44] At step t, Adam stores exponentially decaying averages of gradients m_t and squared gradients v_t:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) \nabla_{\mathbf{W}} C(\mathbf{W}, x_\mathrm{i}, y_\mathrm{i}) \tag{2.11}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) \nabla_{\mathbf{W}} C(\mathbf{W}, x_\mathrm{i}, y_\mathrm{i})^2 \quad , \tag{2.12}$$

with bias values β_1 and β_2.[44] m_t and v_t are termed as averages on first and second moment of the gradient, respectively. After a bias-correction

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \tag{2.13}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t} \quad , \tag{2.14}$$

they are used to compute new weights:

$$\mathbf{W}_{\mathrm{new}} = \mathbf{W} - \frac{\lambda}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t \quad . \tag{2.15}$$

ϵ is an increment for numerical stability.[44]

2.2.3 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) identify, classify and understand important features in the given data input better than fully connected feedforward neural networks.[45] CNNs are based on the **convolutional operation** of a function $f(t)$ with a weighted average operation $w(t)$ denoted by an asterisk. Due to the the discrete nature of (our) data sets, the discretized convolution operation is implemented:[29,32]

$$F(t) = (f * w)(t) = (w * f)(t) = \int f(a) \cdot w(t-a)da \approx \sum_{a=-\infty}^{\infty} f(a) \cdot w(t-a) \quad . \tag{2.16}$$

Here, $f(t)$ is referred to as the **input** data, the weighted average operation $w(t)$ as **kernel** or **filter** and the output $F(t)$ as **activation** or **feature map**.[32,46]

Convolutional Layers

An exemplary convolution operation is shown in Fig. 2.9 where a filter F convolves over an input volume W, thus producing a two-dimensional feature map. After each step of size or **stride** S, the dot product is computed at each spatial position. The number of channels or the depth of the filter has to coincide with the depth of the input volume. Further, the depth of the feature map is equal to the number of applied filters N. If the input volume is not padded with an amount of P numbers - e.g. zeroes - the output size $O = \frac{W-F+2 \cdot P}{S} + 1$ of the feature map (assuming quadratic filters and maps) naturally decreases, which is referred to as **downsampling**.[46,47]

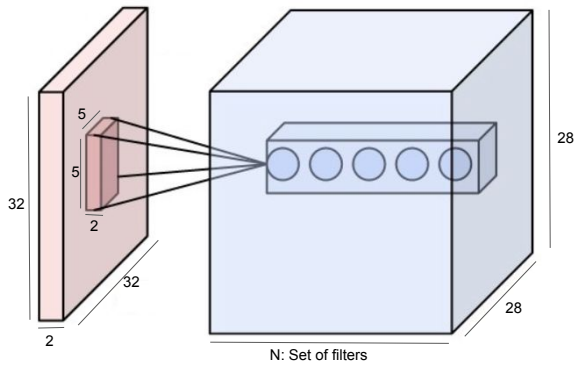


Figure 2.9: A filter ($5 \times 5 \times 2$) convolves over an input volume ($32 \times 32 \times 2$) and computes a feature map ($28 \times 28 \times N = 5$). At each position of the kernel, the dot product of its weights and the input volume is computed. The five neurons (blue circles) illustrate five distinct weights originating from five different filters. Image modified from [48].

All neurons in one feature map share the same parameters (weights) at one filter position. Moreover, they are only connected with some neurons of the input volume. This **local connectivity** allows to detect meaningful features while reducing memory requirements. The **perceptive field** of neurons in one feature map is determined by the filter size F and the stride S.[46,47]

Transposed Convolutional Layers

Contrary to convolutional layers, the main idea behind transposed convolutional layers is to create an output feature map with greater dimensions than the input feature map, as displayed in Fig. 2.10. This **upsampling** can for example be used to reconstruct an original image.[46]

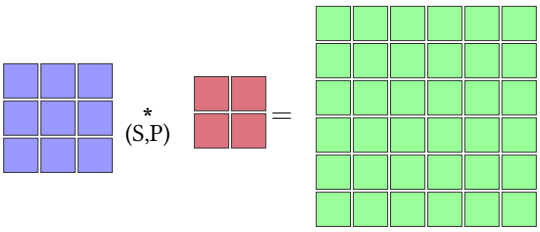
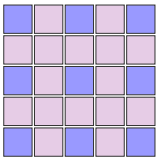
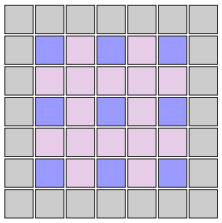


Figure 2.10: Scheme of a transposed convolutional operation. The input feature map (blue) is convolved by a kernel (red) with trainable weights and an output feature map (green) is created. The parameters (S,P) determine the transposed convolution.

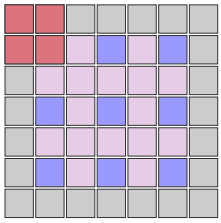
To upsample the input map, it is modified by padding P and zeroes Z, which is illustrated step-by-step in Fig. 2.11. Analogously to standard convolutional layers, the kernel slides over the modified input map with a stride of $S' = 1$ (see Subfigs 2.11c and 2.11d) and computes spatially located dot products (see Fig. 2.11). The output dimension $O = (W-1)\cdot S+F-2\cdot P$ depends thus on kernel size F, input size W, zeroes Z and padding P.[46,49]



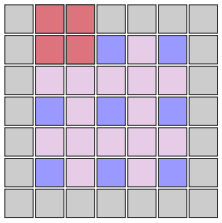
a) First step: The input feature map (blue) is expanded by $Z = S - 1$ zeroes (violet.)



b) Second step: P' zeroes (grey) are padded around the modified input feature map (blue and violet).



c) Third step: Computation of the first dot product with the kernel (red) at position one.



d) Fourth step: Computation of second dot product after stride $S' = 1$ of the kernel (red) to position two.

Figure 2.11: a)+b) Show the modification of the input map and c)+d) the striding of the kernel. Images recreated from [49].

2.2.4 Generative Adversarial Networks

In **deep generative modeling** (DGM), a complex, high-dimensional probability distribution P_X is approximated by training a deep neural network with a large data set. DGMs can not only provide a measure for the affiliation of a sample to the original sample space but also generate new samples that resemble the learned data set.[50–52] **Generative Adversarial Networks** (GAN) constitute a class of DGMs that have been originally introduced in 2014 by Goodfellow *et al.*[53] and were considered "the most interesting idea in the last 10 years in machine learning".[50] A GAN consists of two submodels. The first submodel, known as the **generator** g_θ with parameters θ, is trained to map sample points z from an easier distribution P_Z, i.e. an uniform or Gaussian noise distribution, to a generative probability distribution $P_{X'} = g_\theta(Z)$ such that $g_\theta(z) = X'$ resembles X:[50]

$$g : \mathbb{R}^q \to \mathbb{R}^n \tag{2.17}$$

$$z \mapsto g_\theta(z) \approx x \quad . \tag{2.18}$$

Each sample x in the sample space belongs to an unknown, high-dimensional, intractable probability distribution P_X. X is a statistical property such as the structural configuration of a surface. Since the vectors z are also unknown, they are named as **latent variables** and the probability space P_Z as **latent space** with the statistical property Z.[50] In GANs, the latent space can be sampled with **noise**.[52] Normally, the dimension q of the probability space of P_Z is lower than the dimension n of P_X. An illustration of this mapping can be seen in Figure 2.12.[50]

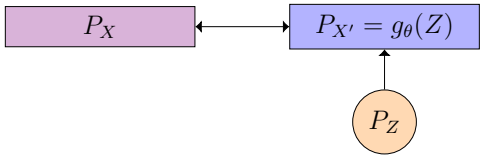


Figure 2.12: A deep generative model g_θ maps the statistical property Z from the easier distribution P_Z to a generative probability distribution $P_{X'} = g_\theta(Z)$ which resembles P_X. Image recreated from [50].

The goal is to train the generator g_θ such that the generated samples $g_\theta(z)$ cannot be distinguished anymore from samples x of the real data space X ($g_\theta(z) \approx x$). To distinguish whether the sample belongs to the real sample space ($x \sim X$) or whether the sample is generated by the generator ($x \sim g_\theta(z)$), a second neural network, the **discriminator** d_ϕ with parameters ϕ, is introduced as a decision maker:

$$d_\phi : \mathbb{R}^n \to \{0, 1\} \quad . \tag{2.19}$$

It functions as a **binary classifier** with $d_\phi(x) = 1$ for $x \sim X$ and $d_\phi(x) = 0$ for $x \sim g_\theta(Z)$. This decision process is illustrated in Fig. 2.13.

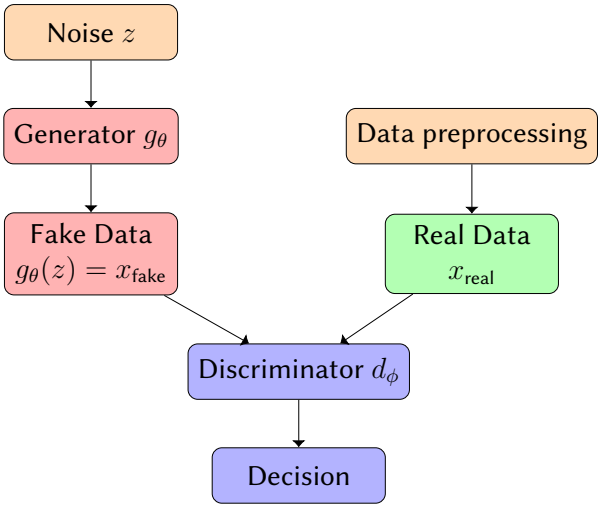


Figure 2.13: A generator g_θ (red) maps noise z (orange) to fake data. The discriminator d_ϕ (blue) distinguishes then between the real data (green) and the generated data (red). Image recreated from [51].

Since the discriminator performs a binary classification optimization problem, we choose the **binary cross-entropy** (see Section 2.2.2) as a loss function:[51]

$$\min_{g_\theta} \max_{d_\phi} \text{CE}_{\text{GAN}}(\theta, \phi) = \min_{g_\theta} \max_{d_\phi} E_{x\sim X}[\log(p(d_\phi(x)))] + E_{z\sim Z}[\log(1 - p(d_\phi(g_\theta(z))))] \quad , \tag{2.20}$$

where $p(d_\phi(x))$ denotes the decision probability of the discriminator that the sample was taken from the real data distribution and $p(d_\phi(g_\theta(z)))$ the decision probability of the discriminator that the sample was generated. The discriminator is trained to maximize the probability of a correct classification, whereas the generator simultaneously attempts to minimize the same quantity.[52] Hence, these probabilities quantify the training process in which the generator learns to create more and more realistic fake data and in which the discriminator conversely has to adapt to better distinguish the fake data from the real data set. The training process can be interpreted as a **two player non-cooperative game** or as a **Nash equilibrium** between the generator and the discriminator and is the reason why the two networks are termed as adversarial networks.[50]

In two successive optimization steps, the parameters of both networks ϕ and θ are updated by the SGD optimizer (see Section 2.2.2 or Eq. (2.10)) with the learning rates λ_ϕ and λ_θ in the k-th step:[50]

$$\phi^{(k+1)} = \phi^{(k)} + \lambda_\phi^{(k)} \frac{1}{s} \sum_{i=1}^{s} \left[\nabla_\phi \log\left(p(d_{\phi^{(k)}}(x_i))\right) + \nabla_\phi \log\left(1 - p(d_{\phi^{(k)}}\left(g_{\theta^{(k)}}(z_i)\right))\right)\right] \tag{2.21}$$

and

$$\theta^{(k+1)} = \theta^{(k)} + \lambda_\theta^{(k)} \frac{1}{s} \sum_{i=1}^{s} \left[\nabla_\theta \log \left(1 - p(d_{\phi^{(k+1)}} \left(g_{\theta^{(k)}}(z_i) \right)) \right) \right] \quad . \tag{2.22}$$

In Eq. (2.21) and (2.22), the discriminator with parameters $\phi^{(k)}$ is optimized in the first optimization step. Consecutively, the generator with parameters $\theta^{(k)}$ is trained utilizing the updated discriminator $\phi^{(k+1)}$. In the literature, there are also other approaches that apply different optimizers (e.g. Adam[44], see Section 2.2.2).[50] For a perfectly trained generator, the generated data $x \sim g_\theta(Z)$ becomes statistically indistinguishable from the real data space for the discriminator[50] such that the average output of the discriminator would be $\frac{1}{2}$. Determining the optimal solution for the weights θ and ϕ is a **saddle point problem**.[50–52]

As illustrated in Fig. 2.13, no reference labelling or categorization of the data is required for the discriminator to perform a classification. GANs are thus an unsupervised learning algorithm.[52] Through the framework of a GAN, it can be avoided to explicitly compute the probability $p_X(x)$ that a sample x belongs to the probability space X:

$$p_X(x) = \int p_g(x|z) p_Z(z) dz \quad , \tag{2.23}$$

where $p_Z(z)$ gives the probability of the vector z in Z and $p_g(x|z)$ denotes the probability for accuracy of the mapping $g_\theta(Z)$ to the sample space X meaning that one specific $z \in Z$ results in one specific $x \in X$.[50] Instead, a GAN tries to reduce the difference between two probability distributions P_X and $P_{X'}$ during the training loop, which is equivalent to minimizing the **Kullback-Leibner** (KL) divergence:

$$\mathrm{KL}(P_X \| P_{X'}) = \int p_X(x) log \frac{p_X(x)}{p(g(z))} dx \tag{2.24}$$

between $p_X(x)$ and $p(g(z))$.[52] However, the KL divergence is asymmetric and could lead to unprecedented training behavior. Thus, the **Jensen-Shannon** (JS) divergence is computed in standard GANs:[51]

$$\mathrm{JS}(P_X \| P_{X'}) = \frac{1}{2} \mathrm{KL} \left(P_X \| \frac{P_X + P_{X'}}{2} \right) + \frac{1}{2} \mathrm{KL} \left(p_{X'} \| \frac{P_X + P_{X'}}{2} \right) \quad . \tag{2.25}$$

After a successful training process, the GAN can be used to generate new samples. In inverse structure design, the outputs of the GAN are used as suggestions for new structures. Since they are assumed to neither be fully converged nor physically meaningful, these structures are given as an input to a geometry optimization to determine if a sensible and previously unexplored structure on the potential energy surface was found[19,20]

Challenges in GAN-Training

An advantage of GANs is that the training process can be parallelized, especially in the case of large data sets.[51] However, the most apparent disadvantage of GANs is the cumbersome, computationally expensive and volatile training process.[50] On top of that, necessary training hyperparameters like learning rates are highly sensitive to modifications of the computational architecture.[54]

One common problem during GAN training are **vanishing gradients**. Arjovsky *et al.* proved that if the two probability distributions P_X and $P_{X'}$ lie on low-dimensional manifolds or have disjoint supports, there always exists a perfect discriminator whose gradients are zero for any generated sample point $g_\theta(z)$. Since the generator is trained through a gradient-based optimization algorithm, vanishing gradients in the discriminator slow down the training process and can even prevent the training from converging completely in the worst case.[51,54]

The strong dependence of GANs on the ratio of the learning rates λ_ϕ and λ_θ goes hand in hand with the previous problem and beyond. If the discriminator learning rate λ_ϕ is too large, it becomes impossible for the generator to improve because the gradient $\nabla_\theta \mathrm{CE}_{\mathrm{GAN}}(\theta, \phi) \approx 0$. If, however, the learning rate λ_ϕ is chosen too small the generator cannot update its weights properly because inaccurately generated data is falsely accepted by the discriminator.[50] The mathematical reason for this instability is inherent to the approach of optimizing parameters of neural networks itself. Goodfellow *et al.* have proven that there exists a unique solution for the ideal generator g_θ and discriminator d_ϕ in **function space** for any probability distributions P_X and $P_{X'}$.[53] However, this guarantee for convergence does not hold during the training process of a neural network because it takes place in **parameter space**.[51]

As introduced in Eq. (2.21) and (2.22), the networks are updated iteratively. This leads to the problem that the solutions of a maxmin and a minmax problem are not necessarily equal:[51]

$$\min_{g_\theta} \max_{d_\phi} \mathrm{CE}_{\mathrm{GAN}}(\theta, \phi) \neq \max_{d_\phi} \min_{g_\theta} \mathrm{CE}_{\mathrm{GAN}}(\theta, \phi) \quad . \tag{2.26}$$

To avoid vanishing gradients, the '$-\log$'-trick is used as an alternative update scheme for the generator in which the part of Eq. (2.22) $\log\left(1 - p(d_{\phi^{(k+1)}}\left(g_{\theta^{(k)}}(z_i)\right))\right)$ is changed to $-\log\left(p(d_{\phi^{(k+1)}}\left(g_{\theta^{(k)}}(z_i)\right))\right)$.[51,54] However, this leads to a loss function that has an extremely low cost on dropping modes for the generator g. Consequently, this will result in the problem of **mode collapse**, where g will most likely not cover the **multi modal distribution** P_X but rather only modes with the highest acceptance rates are learned.[54] Worst case scenario, only a single output x_1 is produced by the generator for any input z:[50]

$$g_\theta(z) = x_1 \quad \forall z \sim Z \quad . \tag{2.27}$$

The phenomenon of mode collapse is worsened by the fact that the generator g lies in the inner training loop of the iterative gradient optimizer update (see Eq. (2.21) and (2.22)). Here, g

aims to generate data that is likely to be accepted by the updated discriminator. Consequently, g often only learns a single mode of the data distribution.[51] In practice, it is easy to detect a mode collapse by comparing samples from the generator - which would all be similar or even identical in the case of a mode collapse.[50]

Another challenge arises due to an upper limit of the divergences during a GAN training process, as it is proven by Theorem 2.3 from [54]:

"Let P_X and $P_{X'}$ be two distributions whose support lies in two manifolds M and P that don't have full dimension and don't perfectly align. We further assume that P_X and $P_{X'}$ are continuous in their respective manifolds. Then,

$$\begin{aligned} \mathrm{JS}(P_X \| P_{X'}) &= \log 2 \\ \mathrm{KL}(P_X \| P_{X'}) &= +\infty \\ \mathrm{KL}(P_{X'} \| P_X) &= +\infty\text{"} \quad . \end{aligned}$$

Therefore, the loss cannot be minimized by means of gradient optimization algorithms. Arjovsky *et al.* thus proposed to introduce a softer measure for the distance of two probability distributions than either JS- or KL-divergence, which will be described in the following.[54]

Wasserstein-GAN

There is a plethora of GAN variants that try to tackle the difficulties of convergence in the training process by introducing further measurements for the distance or divergence between the two probability distributions, e.g. Least squares GAN (LSGAN), Fisher GAN (FGAN) or Wasserstein GAN (WGAN).[51]

The latter approximates the **Wasserstein Distance** or **Kantorovich–Rubinstein Metric** which is also known in computer science as the **Earth Mover's Distance** (EMD):[55]

$$\mathrm{EMD}(P_X, P_{X'}) = \inf_{\gamma \in \Pi(P_X, P_{X'})} E_{(x,z)} \left[\|x - z\| \right] \quad , \tag{2.28}$$

where $\Pi(P_X, P_{X'})$ is the set of all joint distributions $\gamma(x, z)$ and contains all possible pairs of the **marginal distributions** P_X and $P_{X'}$ of the individual random variables X (real data) and Z (latent space data). The Euclidean norm $\|x - z\|$ measures the distance of those two probability spaces. A vivid depiction of the EMD is that it works as a **cost function** for a transport optimization problem. In this picture, the probability distribution $P_{X'}$ can be interpreted as a *pile of soil* that shall be transported into a *hole* that resembles the probability distribution P_X. The joint probability distribution $\gamma(x, z)$ describes how much *dirt* has to be transported from z to x to transform $P_{X'}$ into P_X and the Euclidean norm $\|x - z\|$ measures how far the *dirt* has to be moved. The EMD, as a product of the mass times transport distance, acts as a **cost function** for the transport plan.[50,55,56]

Arjovsky *et al.* proved in [55] that the EMD provides a continuous loss function for the parameters θ that is differentiable almost everywhere in feedforward neural networks with pointwise nonlinearities. This ensures that a continuous mapping $\theta \mapsto g_\theta$ can be achieved in which a sequence of parameters θ_i can converge to θ and thus $g_{(\theta_i)}$ to $g_{(\theta)}$. It is found empirically that this improves the convergence of gradient-based optimization algorithms as well as the sample quality.[55]

Computing the infimum in Eq. (2.28) is tedious because there are infinitely many transport plans to move the *soil*. Therefore, an approximation of the EMD using the Kantorovich-Rubinstein duality[57] is typically implemented in WGANs:

$$\mathrm{EMD}_1(P_X, P_{X'}) = \max_{f \in \mathrm{Lip}(f) \leq 1} E_{z \sim P_{X'}}\left[f(g_\theta(z))\right] - E_{x \sim P_X}\left[f(x)\right] \quad . \tag{2.29}$$

Here, f denotes all Lipschitz-1 continuous functions.[52,55]. If $f \in \mathrm{Lip}(f) \leq 1$ is replaced by $f \in \mathrm{Lip}(f) \leq K$, the set of all K-Lipschitz continuous functions would lead to the EMD up to a multiplicative constant K ($K \cdot \mathrm{EMD}(P_X, P_{X'})$).[55] In WGANs, f is realized as a neural network - the so-called **critic** - which is the adversarial of the generator in the training loop.[52] In contrast to normal GANs, the critic estimates the EMD as parameterized functions $\{f_w\}_{w \in W}$:

$$\mathrm{EMD}_1(P_X, P_{X'}) = \max_{w \in W} E_{z \sim P_{X'}}\left[f_w(g_\theta(z))\right] - E_{x \sim P_X}\left[f_w(x)\right] \quad . \tag{2.30}$$

The generator attempts to minimize Eq. (2.30) to in turn minimize the EMD between the real data distribution P_X and the fake data distribution $P_{X'}$. Hence, the new loss functions can then be rewritten as:[52]

$$\mathrm{EMD} = \min_{g_\theta} \max_{w \in W} E_{z \sim P_{X'}}\left[f_w(g_\theta(z))\right] - E_{x \sim P_X}\left[f_w(x)\right] \tag{2.31}$$

$$\mathrm{EMD}_{\mathrm{WGAN}}(\theta, \phi) = \min_{g_\theta} \max_{d_\phi} E_{z \sim P_{X'}}\left[d_\phi(g_\theta(z))\right] - E_{x \sim P_X}\left[d_\phi(x)\right] \quad . \tag{2.32}$$

In contrast to Eq. (2.20), the new loss function contains no logarithmic dependence and the critic is obviously no longer a binary classifier. Additionally, some of the above described training challenges are resolved as meaningful gradients can be provided at all times for the training process. This simultaneously reduces the probability for the phenomenon of vanishing gradients and mode dropping.[54,55]

In the implementation of a WGAN, the parametrized critic d_ϕ is required to be K-Lipschitz continuous for some K to ensure continuous and meaningful gradients during the whole training process. One way to enforce this is to clamp the magnitude of each weight in each layer in a neural network to a fixed interval, e.g. $[-0.01, 0.01]$, such that the parameters ϕ lie in a compact space.[55] However, this **weight clipping** has some major drawbacks on the training process as too large clipping intervals can increase the training duration whereas too narrow clipping intervals can lead to vanishing gradients.[55]

Therefore, Arjovsky *et al.* proposed another implementation technique in [56] where they prove that a function is 1-Lipschitz continuous if its gradient norm is at most 1 everywhere. To realize this, they introduce a soft version of a **gradient penalty** gp:[56]

$$\mathrm{gp} = E_{x_i \sim P_{\mathrm{gp}}} \left[\left(\|\nabla_{x_i} d_\theta(g_\theta(z))\|_2 - 1 \right)^2 \right] \quad . \tag{2.33}$$

Here, the samples x_i are part of the **sampling distribution** P_{gp} that is formed by a linear interpolation between points x of the real probability distribution P_X and generated points $g(z)$ of the generator probability distribution $P_{X'}$:

$$x_i = \alpha \cdot x + (1 - \alpha) \cdot g(z) \tag{2.34}$$

with a random value $0 \leq \alpha \leq 1$.[56] The gradient penalty is added on top of the EMD loss function multiplied with a constant λ_{gp}, thus leading to a new WGAN loss function $\mathrm{L}_{\mathrm{WGAN}}$:[56]

$$\begin{aligned} \mathrm{L}_{\mathrm{WGAN}} = \min_{g_\theta} \max_{d_\phi} & E_{z \sim P_{X'}} \left[d_\phi(g_\theta(z)) \right] - E_{x \sim P_X} \left[d_\phi(x) \right] \\ & + \lambda_{\mathrm{gp}} \cdot E_{x_i \sim P_{gp}} \left[\left(\|\nabla_{x_i} d_\theta(g_\theta(z))\|_2 - 1 \right)^2 \right] \quad . \end{aligned} \tag{2.35}$$

3 Computational Methodology

In this section, we will first describe how we created two different RuO_2 datasets and how we designed the input for our WGAN framework. We will then proceed to the different implementations of the WGAN architectures. Here, four different approaches to encode additional physical information, such as the energy of the structures or the corresponding lattice lengths, will be described. Our computational design is created in a similar way to [19, 20], who developed a GAN framework for the prediction of novel zeolite structures that takes two input channels: one for an energy grid and one for a materials grid encoding the atomic positions within the zeolites. The codes for all computational architectures are listed in the gitlab repository in [58].

3.1 Dataset design

As described above, GANs as a subclass of generative models learn an underlying unknown probability distribution of a sample space by processing a finite amount of samples representative for that sample space. Our objective is to train a GAN to learn the chemical space of RuO_2 surface terminations. It is thus crucial for the success of our training process to use tailored datasets that cover the chemical space of interest and to create a suitable input format to our neural networks that encodes the desired information. In this thesis, two different datasets have been used. The difference between those two training sets will be highlighted in the following, as well as some reasoning behind adjustments to the input cell geometries. Afterwards, a description of different approaches to encode physical information like the energy range of the input structures is given.

3.1.1 GAP dataset

Gaussian approximation potentials (GAPs) are a subclass of interatomic machine learning potentials of non-fixed form. These GANs are typically trained with a suitable database - consisting of energies and forces from expensive first-principles computations - as well as a local representation of atomic environments.[59,60] A common choice for such a representation is the so-called smooth overlap of atomic positions (SOAP) descriptor.[61] Based on the input data, GAPs are performing a regression task on local quantities analogously to Gaussian Process Regression (GPR) as they are an application of the latter.[59,60] In principle, GAPs model the potential energy surface as a function of nuclear coordinates without describing the electrons explicitly and predict this local quantity, i.e. the energy of a configuration, based on its similarity to other configurations in the database.[18,60] For a detailed introduction on GAPs und GPR, the interested reader is referred to [60] and [59].

In [18], Timmermann *et al.* used an iterative training protocol to train a GAP with low-index rutile (1×1) RuO_2 facets. Computational details on the setup of the GAP, e.g. interatomic potential, loss function, kernels and representations, and the iterative training protocol can be found in [18]. Based on this trained GAP, a dataset of 148 (1×1) RuO_2 surface structures containing all five crystal facets (001), (100), (101), (110) and (111) was created. All five crystal truncation lead to one oxygen-poor, one stoichiometric and one oxygen-rich surface termination as described above. For the (111) surface, an additional super-O-rich termination and an additional stoichiometric termination were sampled.[18] The structures were stored as atom objects from the Atomic Simulation Environment (ASE) containing additional information such as potential energies obtained by DFT.[62]

To map the structures on a suitable grid as input to our GAN, non-orthogonal cells were extended to orthogonal supercells by an in-house code (see gitlab repositiory). In this script, atomic positions within the original cell remain the same, new orthogonal cell vectors are computed and atom motifs of the original cell are repeated within the newly extended system size. The DFT potential energy is adjusted to the increasing number of atoms in the newly expanded cell. To verify that the DFT potential energy of the newly created supercells scales proportionally to the number of added atoms, two single point DFT computations of three randomly picked samples were computed (settings chosen as in [18]) and only lead to small numeric deviations of $20\,\mathrm{meV}$. In our GAN framework, we aimed to encode the energy for our structures. For this purpose, we did not use absolute energy values but rather energy ranges. Based on the small numeric deviations, we thus assumed that the energies of the expanded cells scale accurately enough to the system size. The energy ranges for the energy encoding were defined as listed in Table 3.1. The DFT energies of the samples were grouped in ten intervals and each interval was then assigned a class label from 0 to 9 which we will refer to as the energy class label.

Table 3.1: The energy range per atom for each energy class label is listed. This classification is applied to both GAP datasets.

Range of energy per atom [meV]	Class label
$[-1220.2, -1201.4]$	0
$[-1201.3, -1182.6]$	1
$[-1182.5, -1163.7]$	2
$[-1163.6, -1144.8]$	3
$[-1144.7, -1125.9]$	4
$[-1125.8, -1107.0]$	5
$[-1106.9, -1088.1]$	6
$[-1088.0, -1069.3]$	7
$[-1069.2, -1050.4]$	8
$[-1050.3, -1031.4]$	9

By visual inspection of the dataset, we found 51 structures containing a broken layer structure with gaps between different RuO_2 layers that had either peroxide groups associated with

the surface or single oxygen atoms floating in the cell, as demonstrated in Fig. 3.1. To inspect the training results in dependence on the structural variety of the dataset, we split the GAP dataset in two subsets, one containing all 148 structures and one only containing 91 structures free of any defects.

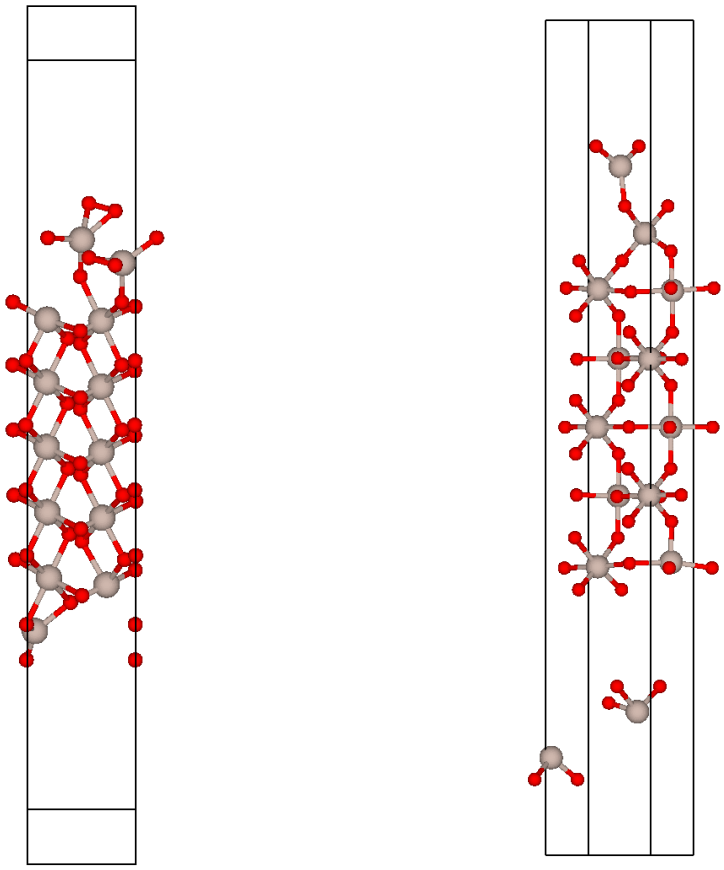


Figure 3.1: Two examples for non-physical structures contained in the first GAP dataset. The left structure contains multiple floating single oxygen atoms. Additionally, the upper structure fragment starts to detach. The right structure has a broken slab structure with disconnected layers. Ru atoms are drawn as grey spheres and O atoms as red spheres.

To highlight the impact of the structure removal from 148 to 91 structures, both sample distributions are plotted in the energy histogram in Fig. 3.2. Even though 51 structures were removed, the full range of the energy spectrum is still covered in the reduced training set.

In a second step, both sets of (1×1) RuO_2 surface cells were extended in x and y direction to obtain a more quadratic cell appearance, ultimately improving the data sampling in 2D (see below). Concomitantly, the vacuum layer was reduced from 20 Å to 12 Å to reduce the compression of the cell in z direction in our mapping process for the data sampling, as the z cell vector was roughly twice the length as the x or y cell vectors for all cells. In the supporting information of [18], the numerical stability of the vacuum layer reduction was explored for RuO_2 and can thus be performed safely. As we want to process the GAN output

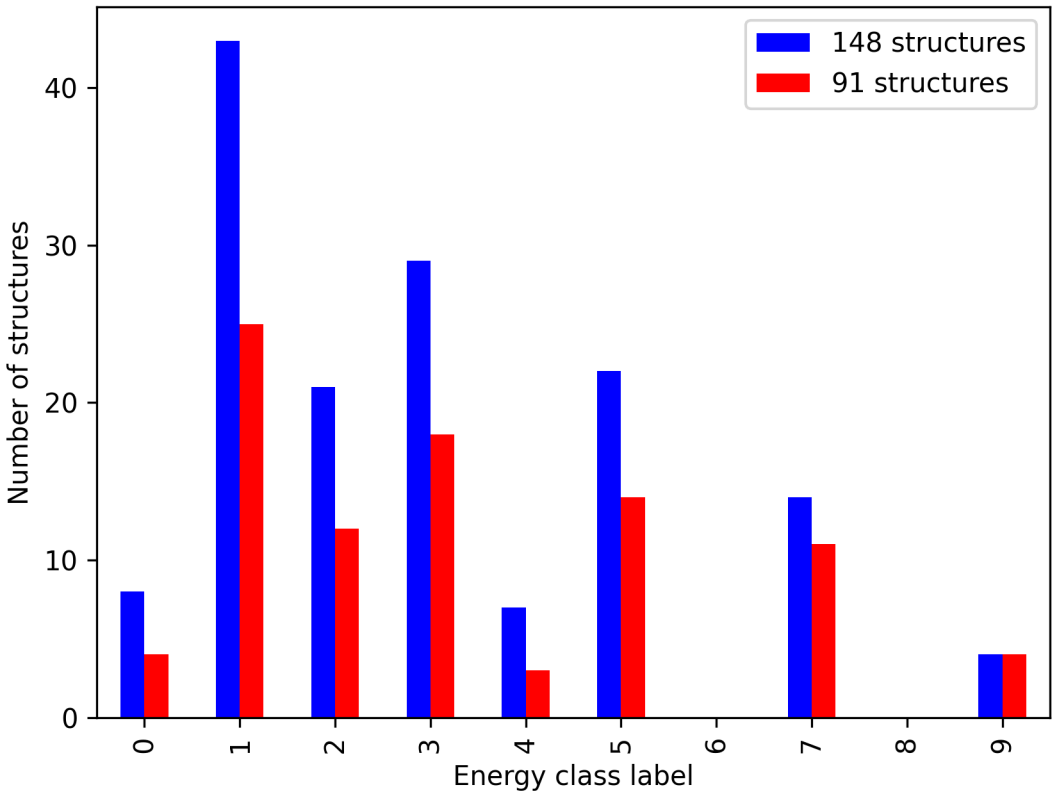


Figure 3.2: Energy histogram showing the amount of structures in each energy class for both GAP subsets.

in later geometry optimizations based on GAP energetics, we aim to directly include a realistic vacuum-slab-layer ratio within a stable numerical range. For one structure, this cell increase and concomitant vacuum layer decrease is illustrated in Fig. 3.3. All extended cells still contain > 2000 atoms.

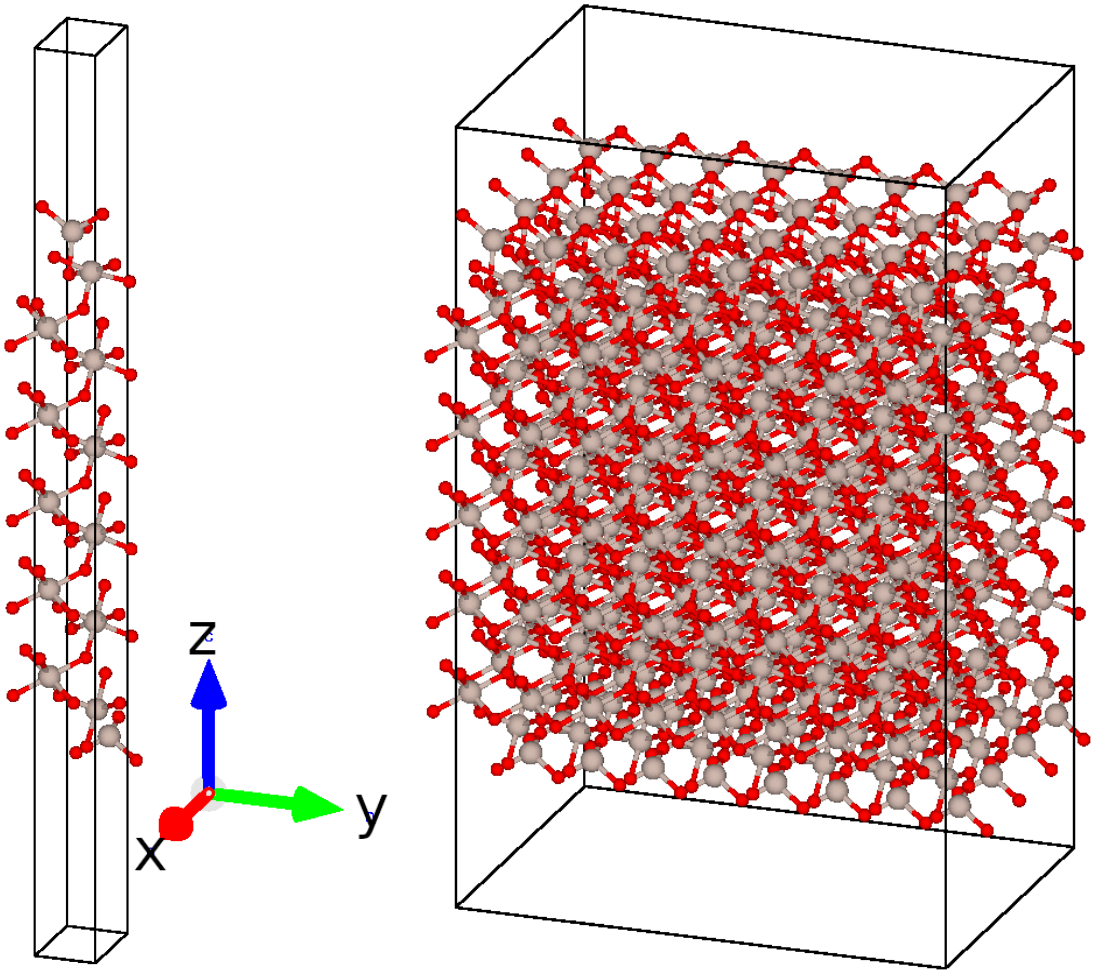


Figure 3.3: Demonstration of the second modification step of the GAP dataset. The original cell (left) is multiplied in x and y cell direction to obtain a quadratic cell (right). The height of the vacuum layer is reduced to 12 Å. Ru atoms are drawn as grey spheres and O atoms as red spheres.

3.1.2 Basin hopping dataset

The second dataset was created with a grand-canonical basin hopping method in Delocalized Internal Coordinates (DICs, also often referred to as curvilinear coordinates) as developed by Panosetti *et al.* in [22, 23] using the trained GAP from [18] for the energetics. DICs are linear combinations of internal coordinates constructed as singular value decomposition of the

redundant Cartesian-internal transformation matrix. It has been shown that the application of displacements in DICs generates more chemically meaningful trial structures than Cartesian displacements, especially for covalent systems, by facilitating the preservation of favourable bonding patterns throughout the sampling. The starting points were a (1×1) $RuO_2(100)$ surface with oxygen-poor and oxygen-terminated stoichiometric termination and a $c(2\times2)$ $RuO_2(100)$ surface with oxygen-poor and oxygen-rich termination, both appropriately repeated to obtain a roughly cubic supercell with dimension of ca. $18\,\text{Å}$. Lattice parameters were taken from the DFT computations of [18]. All cells contained about 2000 atoms and were divided into a displacement layer, a bulk layer and a buffer layer. Both the displacement and the buffer layer cover two Ru layers and and connected oxygen atoms. The DIC displacement was only applied to the displacement layer while the buffer layers is not involved in the displacement but allowed to relax. Atoms in the bulk layer were kept fixed. Displacements of $1.20\,\text{Å}$ were constructed using $25\,\%$ of the available curvilinear modes. Moreover, the removal or insertion probability for oxygen was also $25\,\%$. Oxygen potentials $\Delta\mu_{O_2}$ of $-1.0\,\text{eV}$, $-0.5\,\text{eV}$, $-0.25\,\text{eV}$ and $0.0\,\text{eV}$ were sampled. A Metropolis criterion with a pseudo-temperature of $1000\,\text{K}$ was used. An additional criterion was implemented to reject structures with molecular oxygen or clustered oxygen atoms. The geometry of each trial move was locally optimized using the FIRE algorithm.[63] Each structure and snapshot of the optimization was saved leading to a database of $28,903$ structures. One example for a structure in this dataset is shown in Fig. 3.5. All structures in this dataset are orthorhombic due to the chosen initial cell geometry.

The energy of each structure was computed with the GAP from [18]. Similar to the GAP dataset, all structures were assigned an energy class label from 0 to 9, as shown in Table 3.2. Histogram 3.4 shows the energy distribution over these samples. Since the probability for acceptance was only $25\,\%$, a majority of the structures is located in two energy classes (6 & 7). However, a few hundred samples were located in other energy classes which ensures that the GAN also processed structures from other energy classes during the training process and learned to extract their specific features.

Table 3.2: The energy range per atom for each energy class label is listed. This classification was applied to both GAP datasets.

Energy range [eV]	Class label
$[-520021.6, -519387.3]$	0
$[-519387.2, -518753.0]$	1
$[-518752.9, -518118.7]$	2
$-518118.6, -517484.4]$	3
$[-517484.3, -516850.0]$	4
$[-516849.9, -516215.7]$	5
$[-516215.6, -515581.4]$	6
$[-515581.3, -514947.0]$	7
$[-514946.9, -514312.7]$	8
$[-514312.6, -513678.3]$	9

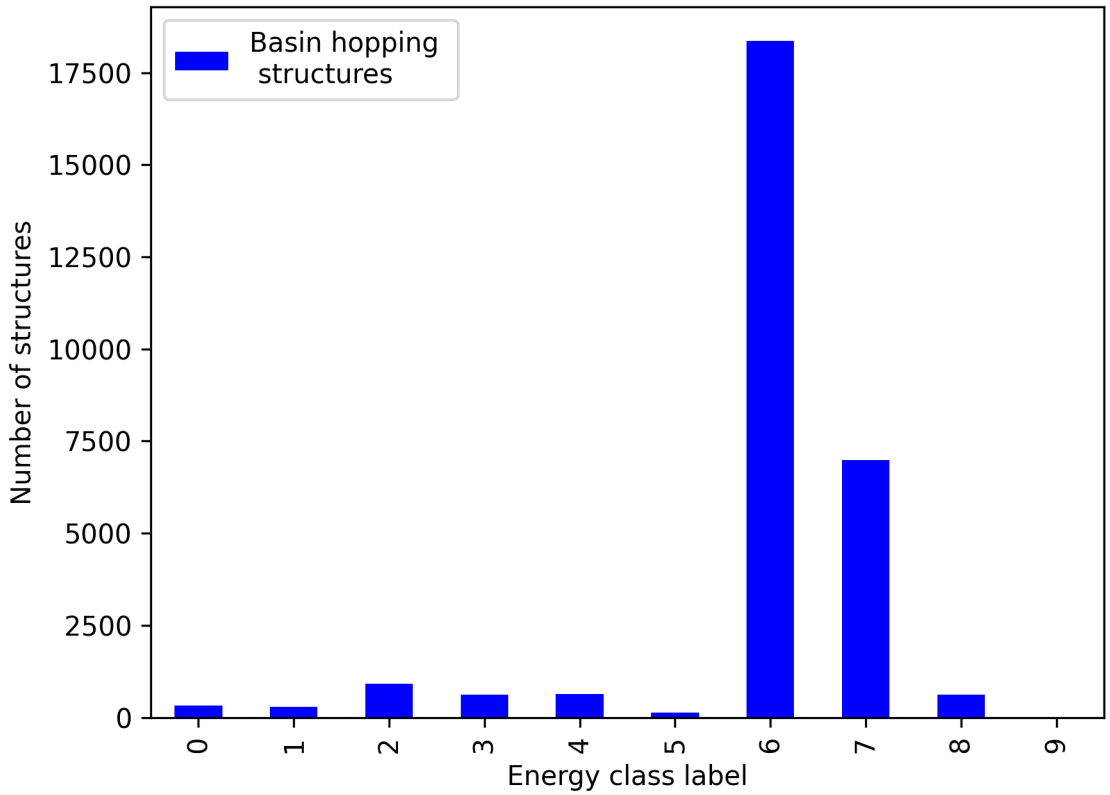


Figure 3.4: Energy histogram showing the amount of structures in each energy class for the basin hopping dataset.

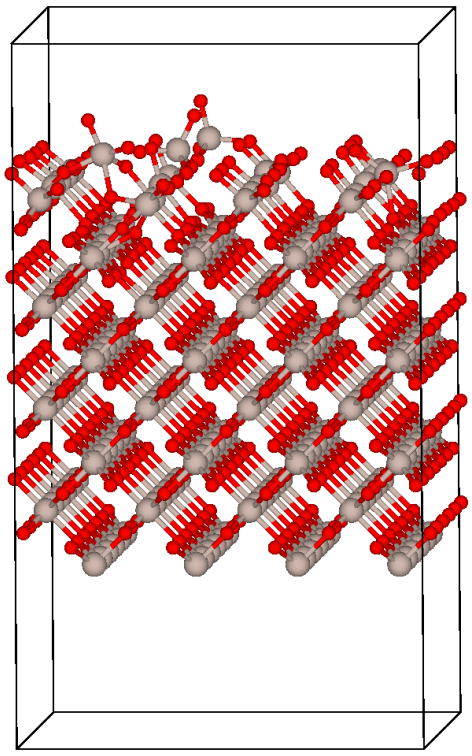


Figure 3.5: One example structure of the dataset created with basin hopping. The majority of the structures is located in two energy classes (6 & 7).

3.2 Atom density-based structural GAN input

The previous section was centered around the creation of two RuO_2 datasets. In these structures, each atomic position is characterized as a distinct point in a unit cell represented by three lattice cell vectors. Here, we describe how this spatial information is converted into a suitable geometric input to the GAN.
During this approach, the atomic positions were encoded as Gaussian shaped pixel intensities on a three-dimensional grid. The main idea behind this approach was that discrete spatial data positions are mapped to a continuous representation. This conversion is crucial because the generator cannot output discrete data in the general case.[52,64] In the learning process, the generator can adapt the intensity of each voxel on the interval $[0, 1]$ and can thus improve the sample quality based on feedback from the critic by making small trial steps and thus adjusting each voxel value individually.
As initial step of the mapping process, we constructed a 32x32x32 grid with orthogonal grid vectors. To facilitate the mapping from the structural data files to our density-based input, the input cells were transformed into orthorhombic cells as described above, in case they were non-orthorhombic beforehand. Here, each atomic position is represented by a Gaussian function centered on the original atomic position in the cell. Consecutively, a three-dimensional stencil was created based on the original atomic position and placed on the 32x32x32 grid. The intensities of two neighboring Gaussian function accumulate if their stencils overlap. One advantage of this representation is that we can already encode some structural flexibility on the atomic positions by adjusting the width of our Gaussian shaped stencils. The width for the Gaussian stencils was controlled as a hyperparameter α and determined the features of our input to the GAN. To guarantee an atomic resolution of at least five voxels in each direction for each Gaussian stencil, each unit cell did not contain more than 2000 atoms. For both atom types (Ru and O atoms), a separate 32x32x32 grid with Gaussians stencils was computed. The created density-based grids were handed to the GANs as separate channels. The mapping was performed using an in-house code as found in the gitlab repository.[58]

3.2.1 2D Sampling

In the first step of the GAN implementation, we started by feeding two-dimensional pictures as input to our network. Since only two-dimensional images are passed through the neural network structures, the amount of required neurons and computations within the networks is reduced by the power of three. This saved computing time during tests of computational architectures and hyperparameters.

To obtain two-dimensional images from the 32x32x32 grid, we sliced the grid along the xz- and the yz-plane as demonstrated in Fig. 3.6 exemplary for a 4x4x4 grid. For each element, the separate 32x32x32 grid is created and sliced.

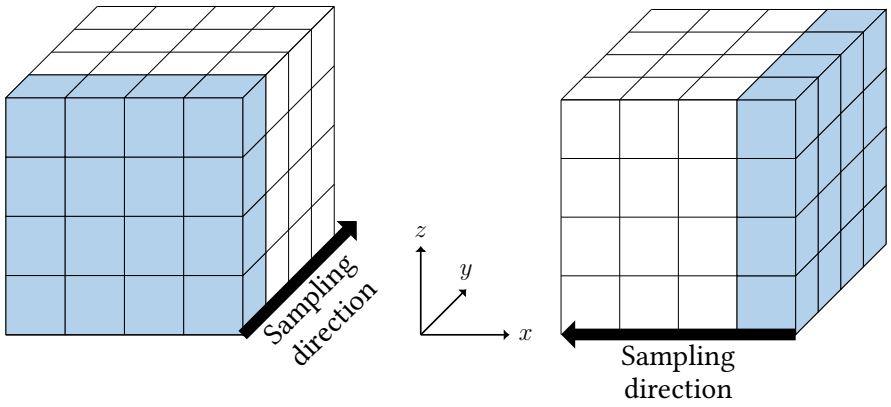


Figure 3.6: Slicing scheme of the density-based grid in two dimensions. The grid was sliced both along the xz- and the yz-plane.

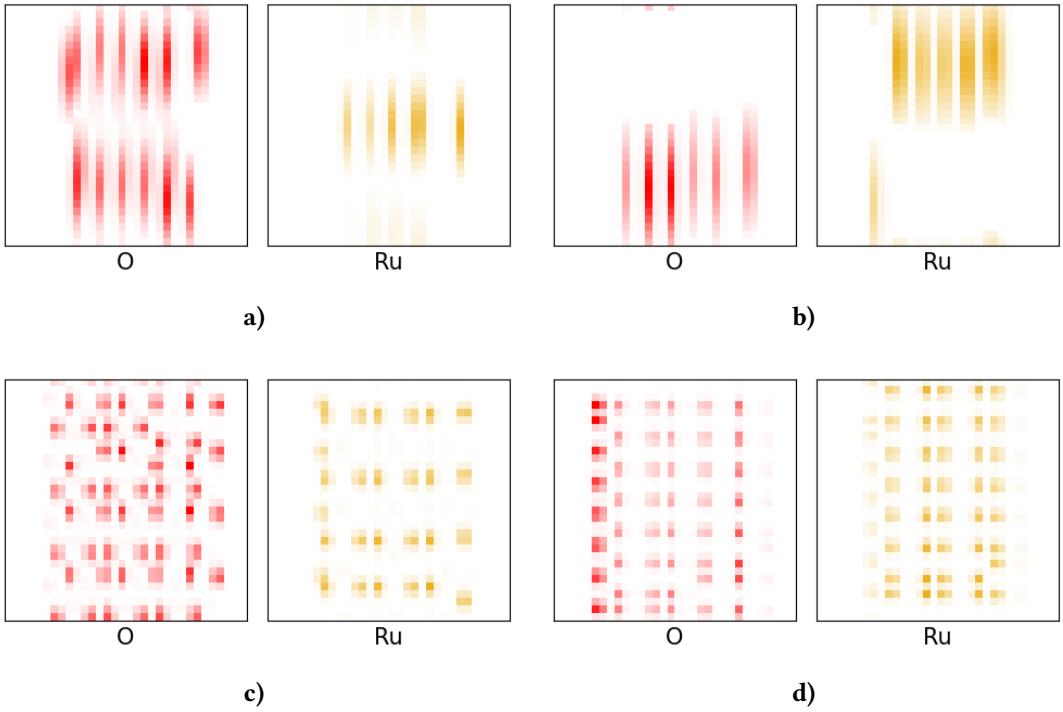


Figure 3.7: Two-dimensional slices of the RuO_2 structures from Fig. 3.3 after the Gaussian mapping to density-based structures. **a)** and **b)** show slices of the original structure in the xz- and yz-plane, respectively. **c)** and **d)** are slices in the corresponding xz- and yz-plane after the cell extension in x- and y-direction.

In Fig. 3.7, some exemplary slices of the structure displayed in Fig. 3.3 are shown before and after the cell extension in x- and y-direction. One can clearly see that before this extension, due to the mapping of all three cell dimensions on the same grid distance, the Gaussians in z-direction are in closer proximity than in x- or y-direction. After the cell extension, the images contain more Gaussian functions representing atoms. As described in the previous section, the

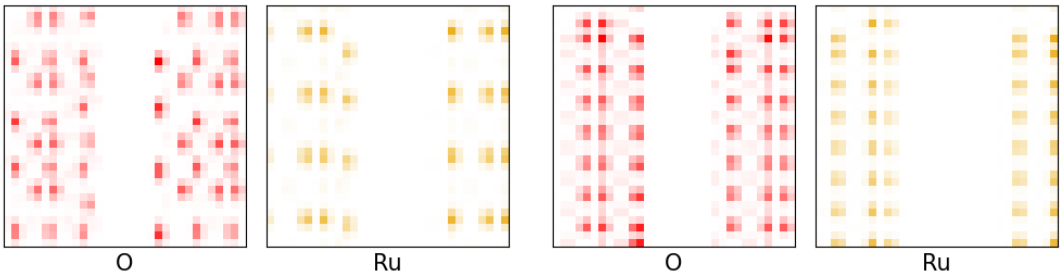


Figure 3.8: a) and **b)** show the randomly translated cell from Subfigs. 3.7c and 3.7d in the xz- and yz-plane, respectively.

width of the Gaussians steers the amount of pixels the atomic positions are represented with. This hyperparameter α was set to $\alpha = 2.5$ and $\alpha = 5.0$ for the smaller and enlarged cells, respectively. Concomitantly to the cell extension, the vacuum region was reduced to increase the resolution of the image area mapping. During the sampling process, empty slices containing no Gaussians were removed. This filter only had to be applied in the two-dimensional application.

To further increase the chemical space covered by our dataset, we introduced random translation and rotation on the three-dimensional grids before the slicing. So far, the vacuum layer was only placed to the left and right of the structures, however, it can also be located in the middle of the cell and the slab can be separated, as seen in Fig. 3.8. Including such operations implicitly taught the GAN periodic boundary conditions in all three cell dimensions. The main reasoning behind a random translation of the dataset in x, y and z direction, however, was to vary the positions of the atomic densities in the cell such that GAN rather learns to map the environment of the atomic densities than to position them at specific spatial locations on the grid.

Each structure in the GAP dataset was randomly translated and rotated four times and then sliced 16 times on even numbered positions on the grid in both x- and y-directions. This resulted in overall respective training set sizes of 18,944 and 11,648 structures for the 148 structures in the GAP dataset ($4 \times 16 \times 2 \times 148$) and the 91 structures in the reduced GAP dataset. The rotations were only executed as multiples of $90°$ along the z-axis to maintain the surface orientation of our density-based inputs.

For the basin hopping dataset, each structure was randomly translated once. In order to recognize overfitting of the GAN, the randomly translated basin hopping dataset was saved and compared to generated data. For the two-dimensional density-based representation (in both x- and y-direction), only one slice was taken at position 10. The final dataset size thus resulted in 57,806 structures ($28,903 \times 2$).

3.2.2 3D Sampling

As described in the previous section, all computational architectures were tested in two-dimensions before passing them into a three-dimensional GAN framework. Analogously to the 2D Sampling, seperate 32x32x32 grids were constructed for Ru and O. Both grids were used directly as three-dimensional input to the neural network. In Fig. 3.9, the extended structure from Fig. 3.3 is plotted with an in-house script represented by the density-based 32x32x32 grid. The GAP datasets were not further used for these computations because they only consists of 148 or 91 structures. Rather, the three-dimensional, once randomly translated basin hopping dataset with 28,903 was used as an input dataset.

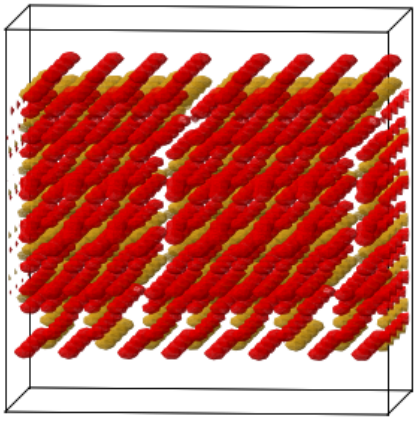


Figure 3.9: Three-dimensional plot of the extended structure from Fig. 3.3 after the Gaussian mapping of atomic positions. Ru atoms are drawn as yellow spheres and O atoms as red spheres.

3.3 WGAN Implementation in PyTorch

The aim of this thesis is to develop and test a deep convolutional WGAN architecture. In this section, we will thus give a brief overview of the code routines and present the used computational architectures to encode energy and lattice information of our chemical systems. The weights in all simulations were automatically initialized by PyTorch using the Kaiming Initialization procedure, as it is described in [65]. All computational architectures for the two-dimensional case were written as single-node single-GPU codes. After increasing our dataset, we upgraded these codes into single-node multi-GPU codes first with the DataParallel[66](DP) PyTorch package and afterwards with the DataDistributedParallel[67](DDP) PyTorch package for efficiency reasons. The main difference between the DP and the DDP package is the implementation of the GPU usage before and after each training epoch and the resulting difference in runtime. The DP package copies the model and data before and after each epoch to the GPU and back to compute averaged gradients before the model update is performed. Contrary to the DP package, the DDP package places a copy of the model to each allocated GPU in the beginning. Only the batched data is copied between epochs, i.e. the models are permanently stored on the respective GPUs and are updated using gradient synchronization

between the models located on different GPUs which leads to a significant speed up. To ensure reproducibility, the CUDA *torch.manual* seed was set to 100 in all DDP codes. The computations were performed on Nvidia A100 NVlink GPUs with 40 GB HBM2 and a CUDA compute capability 8.0/Ampere on the Raven cluster of the Max-Planck computing and data facility (MPCDF).

3.3.1 *Vanilla* 2D-DCWGAN

As described in Section 2.2.4, the Wasserstein-GAN (WGAN) offers an elegant solution to stabilize the volatile GAN training process by introducing a softer measure for the loss function: the Earth mover distance. Hence, we implemented a WGAN in our PyTorch[68] code.[58] As explained in Section 2.2.3, CNNs are better at extracting distinctive and important features than fully-connected layers and have furthermore proven to speed up the convergence in the GAN training process as **Deep Convolutional GANs** (DCGANs).[19,20,69] In DCGANs, the generator is built with transposed convolutional layers for upsampling (see Section 2.2.3) and the discriminator with convolutional layers for downsampling (see Section 2.2.3). Analogously to Kim et al. in [20], we implemented a DCWGAN framework combining the advantages of the EMD as a weaker, converging metric and a computational architectures based on (transposed) convolutional layers facilitating the training process. We used the hyperparameters from Kim et al. as starting points for our hyperparameter searches.[20]

In the following, the general code scheme and computational architecture are introduced for the training process of the DCWGAN that aimed at creating two-dimensional images which either resemble the GAP or the basin hopping dataset. As we only passed our density-based images as inputs to the WGAN and no supplementary information was encoded in further channels or layers, this architecture is referred to as a *Vanilla* WGAN.

The general programming scheme for the *Vanilla* WGAN training process is demonstrated in Fig. 3.10. In our PyTorch code, there is an inner and an outer training loop. First, we will describe how the critic is updated five times in the inner loop and then proceed to the generator update in the outer training loop.

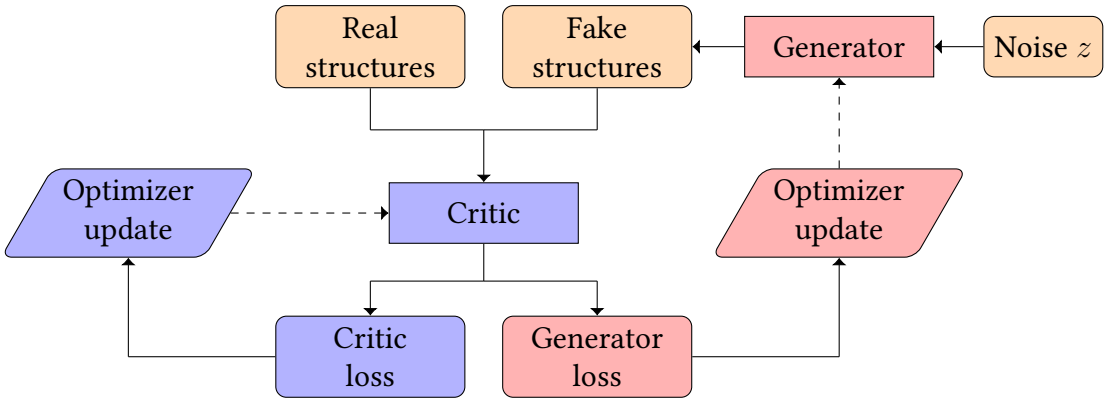


Figure 3.10: Flowchart of the code routine for one iteration of the *Vanilla* WGAN training process.

In the beginning of the inner training loop, the generator is fed randomly sampled noise vectors and produces fake data output, as described in Section 2.2.4. The critic then processes both real and fake structures and computes a loss for both, the decisions on the real $d_\phi^k(x)$ and the decisions on the fake structures $d_\phi^k(g_\theta(z))$. Since we implemented a WGAN, the gradient penalty gp was computed on top for some linearly interpolated samples x_i (see Eq. (2.34)) as described in Eq. (2.33). The value α was chosen randomly in each epoch using the *numpy.random.rand* function. No specific numpy seed was set. The parameter λ_{gp} of the WGAN loss $\mathrm{L}_{\mathrm{WGAN}}$ (see Eq. (2.35)) was set to 10 (value from [55, 56]) for all simulations in this thesis. The WGAN loss $\mathrm{L}_{\mathrm{WGAN}}$ and the critic gradients after backpropagating were passed to the optimizer to update the critic's parameters. For all simulations, the ADAM optimizer (see Section 2.2.2) was initialized for the critic with the bias values $\beta_1 = 0.5$ and $\beta_2 = 0.9$.[44] The critic learning rate λ_{d_ϕ} was varied between 0.001 and 0.0001 throughout the simulations. This training step was repeated five times before the outer training loop proceeded.

In the outer training loop, the generator is updated. Analogously to the inner training loop, noise was sampled and fake structures were generated and passed to the critic to compute the EMD. Accumulated over a whole batch, these EMDs represent the generator loss which is passed on to the generator optimizer together with the gradients after backpropagation of the generator. As for the critic, an ADAM optimizer (see Section 2.2.2) was initialized for the generator with the same values as the critic.[44]

The computational architectures for the critic and generator of the code scheme in Fig. 3.10 are illustrated in Fig. 3.11. As already described, transposed convolutional layers were used in the generator to upsample from the latent space to the 32x32 grid (i.e. to create fake structures). Contrary to that, convolutional layers were used in the critic to downsample the fake or real structures to the EMD. Since the input grid was two-dimensional, two-dimensional convolutional and transposed convolutional layers were required. The batchsize B varied depending on the dataset and the simulation settings between 32, 128, 256, 512, 1024 and 2048.

For the critic, a layer normalization followed by a Leaky-RELU activation function was applied after each convolution with exception of the last convolution operation. To reduce overfitting, connections between neurons were randomly omitted with a probability of $p = 0.5$ after the first convolutional layer. By applying this regularization method, a learning interdependence between the neurons was avoided. The kernel size K, the stride S and the periodic padding P were chosen as depicted in Fig. 3.11 for all simulations.

For the generator, batch normalization was applied to the output of every transposed convolutional layer. Afterwards, a LeakyRELU activation function utilized, with the exception of the last transposed convolutional layer where a sigmoid activation function was employed after the batch normalization. The kernel size K and the stride S were chosen as depicted in Fig. 3.11 for all simulations. No padding was employed. The dimension N of the latent space was either chosen as 512 or 1024.

The *Vanilla* 2D-DCWGAN architecture was tested with the GAP and basin hopping datasets

and computations were performed in the single GPU, the DP multi GPU and the DDP multi GPU code version.

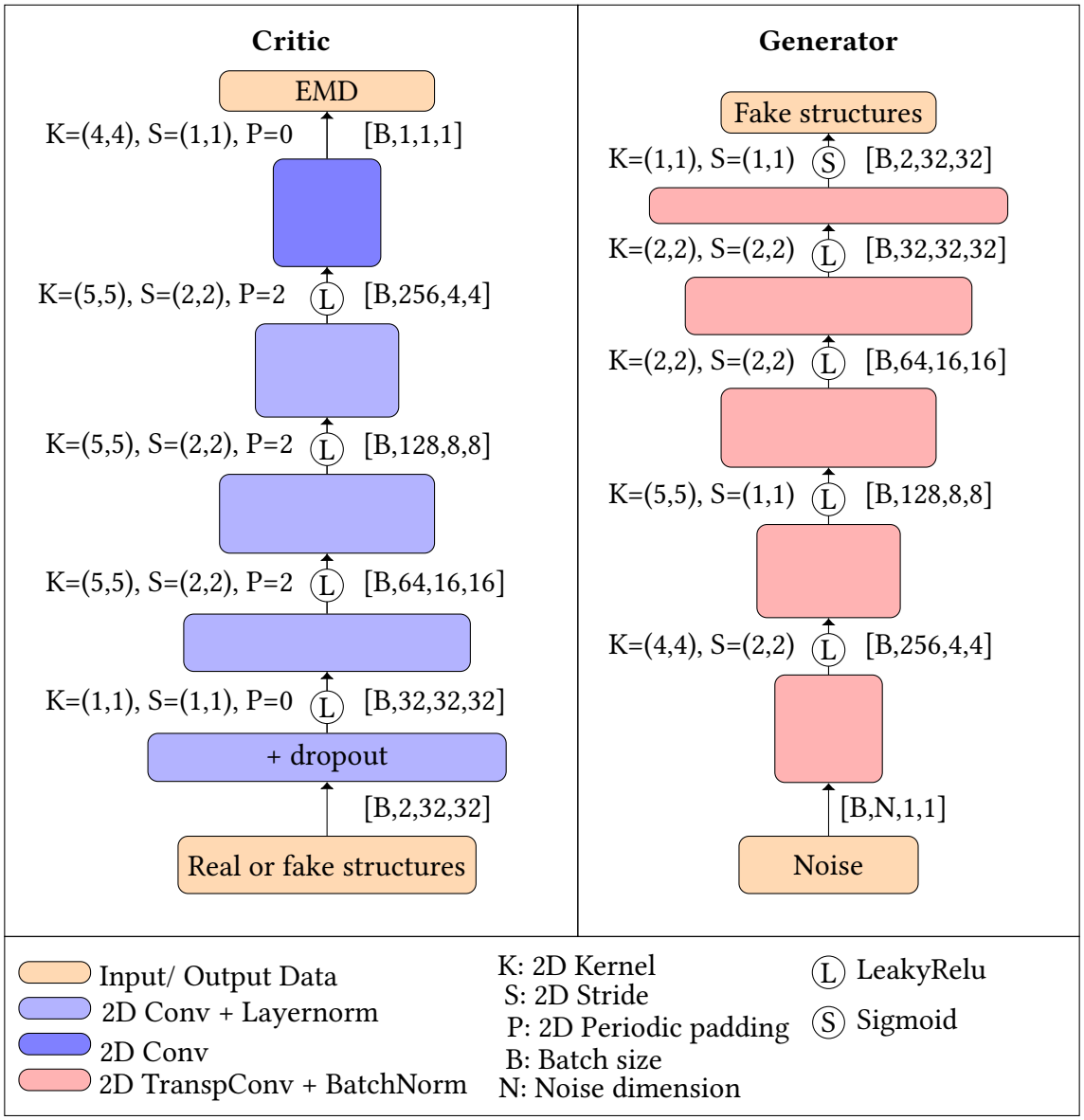


Figure 3.11: Computational architecture and computational details of the implementation of the *Vanilla* 2D-DCWGAN.

3.3.2 Energy encoding in 2D-DCWGAN

In the previous simulation, computational details were presented of a 2D-DCWGAN that is able to learn solely the geometric information of the density-based structures. In this section, however, we want to introduce energy information in our WGAN via data labels. Labels are widely used for data classification[70, 71], for example in the Information Maximizing Generative Adversarial Network (INFOGAN).[72] Here, a DCGAN with structured latent variables was proposed that was trained on the MNIST database - a database of 2D images of handwritten numbers 0 to 9. In addition to the noise vector z, a latent code c consisting of labels 0 to 9 representing the numbers on the pictures was provided as an additional input for the generator.[72] After a successful training process, the generator was then capable to selectively output handwritten numbers from 0 to 9 depending on the concatenated label that was appended to the noise vectors z.[72]

We want to use this idea to encode the energy classes of the structures in our latent space design. For this, the energy labels from Table 3.1 were used for the GAP dataset and the energy labels from Table 3.2 for the basin hopping data set. Thus, the energy labels were 0-9 in both cases. To ensure convergence in the training process, the INFOGAN introduced an additional regularization term in the loss function which is implemented as an auxiliary network.[72] We, however, sought to include the additional latent code in the training process by showing the critic the energy information in a third channel. The code scheme in Fig. 3.12 illustrates how the latent code was provided to the generator and to the critic. Apart from this additional information, the training routine was the same as for the *Vanilla* DCWGAN, as depicted in Fig. 3.10. To access the best energy labels for our problem, the energy information was once encoded as one hot vectors, once as integer-based conditioning vectors and once as an energy embedding. The computational details of each approach are explained in the following.

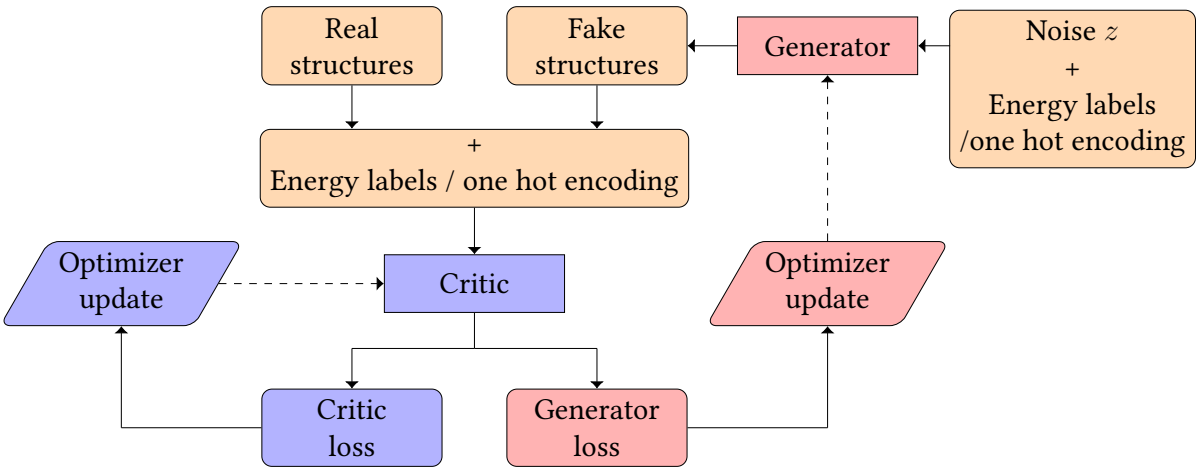


Figure 3.12: Code scheme for the encoding of additional energy information during the training process either in the form of labels or one hot encoded vectors in a 2D-WGAN.

Integer-Based Conditioning Vectors

The idea of energy conditioning is to use the labels 0 to 9 and map them to a conditioning vector of size C that consists of a repetition of the corresponding class label. The size of this integer-based conditioning vector C is thus a new hyperparameter for the simulation. In Fig. 3.13, the computational details are illustrated. Basic computational architectures of Fig. 3.11 and all entailed hyperparameters remain identical. The only deviation is the input to both neural networks.

As described above, the conditioning vector is concatenated to the randomly sampled noise vector z of size N such that the input to generator is of dimension [B,N+C,1,1]. C was varied between 100, 206 and 512 for the generator input. We sampled C in the ratio of $1:4$, $1:2$ and $1:1$ to the size of the noise vector N to explore the impact of the latent code on the latent space design. For the critic input, however, the dimension of the integer-based conditioning vector has to be fixed to [B,1,32,32], such that we can concatenate it as an separate channel to the first convolutional layer.

Computations for the computational architecture of the 2D-DCWGAN with integer-based energy conditioning were only performed with both GAP datasets in the DP multi GPU code version.

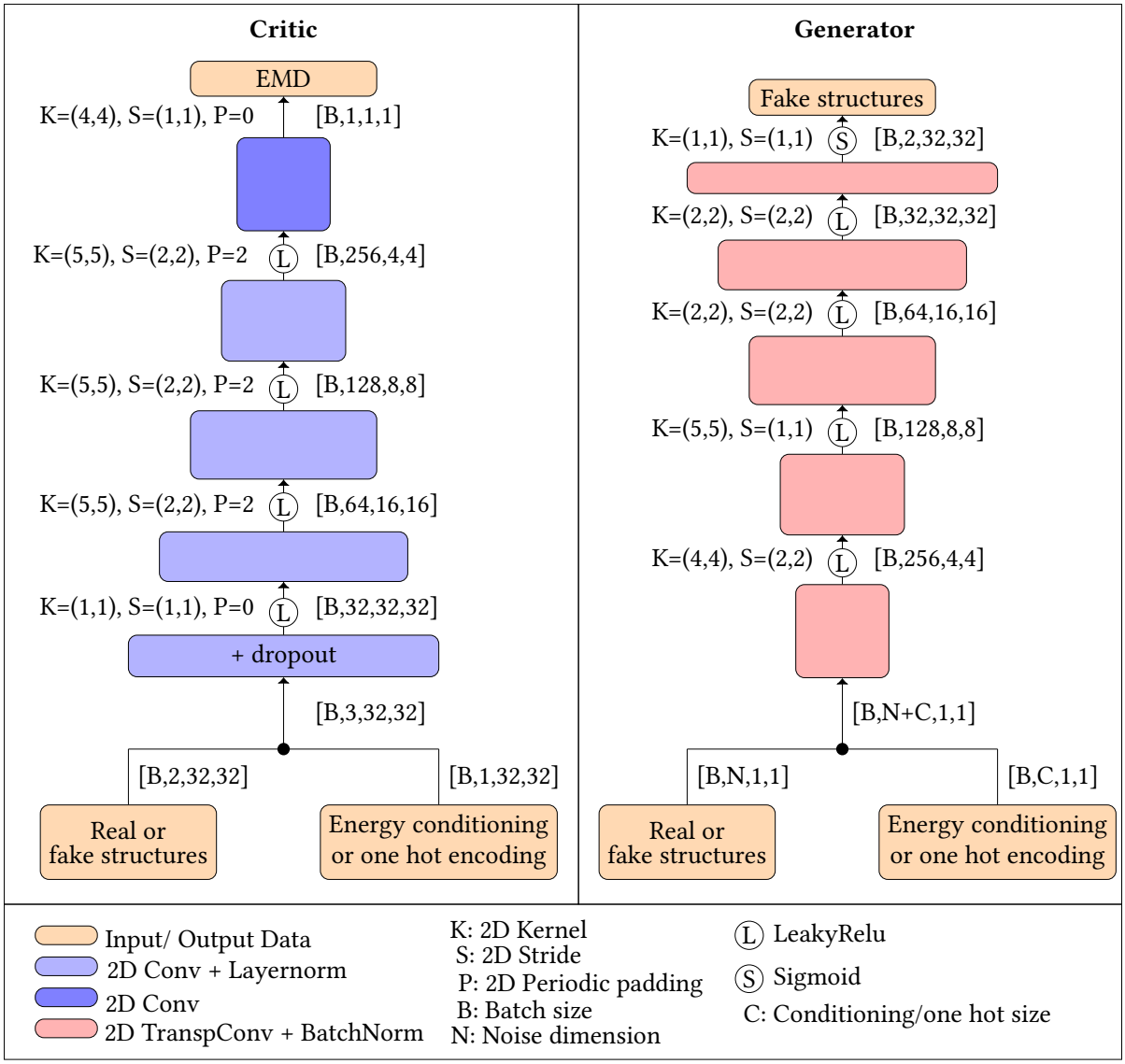


Figure 3.13: Computational architecture and computational details of the implementation of the 2D-DCWGAN with integer-based energy conditioning or one hot encoding.

One Hot Encoding Vectors

The energy information is encoded in one hot encoded vectors of the class labels 0 to 9 in binary representations. An example of a one hot encoding of the labels 0 to 9 for a one hot vector size of 10 is illustrated in Table 3.3. An identical computational architecture as in Fig. 3.13 and identical concomitant hyperparameters were used for these simulations. One hot vectors of size 100, 200 and 500 were concatenated with the noise vector of size N and then provided as input for the generator. Similarly to the previous simulation, the size of the one hot encoding vector was sampled in comparison to the noise dimension N, however, the one hot vector size was given in multiples of 10 due to the choice of 10 class labels. The input size of the one hot vector for the critic was fixed to 32x32 and then resized to [B,1,32,32], similar to the conditioning vector. Computations for the computational architecture of the

2D-DCWGAN with one hot encoded energy conditioning were only performed with both GAP datasets in the DP multi GPU code version.

Table 3.3: One hot encoding of the labels 0 to 9 with a one hot vector size of 10.

Class label	One hot encoding
0	$(1,0,0,0,0,0,0,0,0,0)$
1	$(0,1,0,0,0,0,0,0,0,0)$
2	$(0,0,1,0,0,0,0,0,0,0)$
3	$(0,0,0,1,0,0,0,0,0,0)$
4	$(0,0,0,0,1,0,0,0,0,0)$
5	$(0,0,0,0,0,1,0,0,0,0)$
6	$(0,0,0,0,0,0,1,0,0,0)$
7	$(0,0,0,0,0,0,0,1,0,0)$
8	$(0,0,0,0,0,0,0,0,1,0)$
9	$(0,0,0,0,0,0,0,0,0,1)$

Energy Embedding

Energy embedding presents an additional network layer that is augmented on both the critic and the generator to encode the integer labels 0 to 9, as demonstrated in Fig. 3.14. The additional layer constituted of an embedding layer for the generator and of an embedding layer followed by a linear, fully-connected layer for the critic to reshape the input accordingly.

The concept of an embedding layer is to embed the given labels in a larger vector space of dimension E with an amount of N_e embedding classes and to simultaneously create an accessible reference table. This entails that each vector is assigned an index. In the beginning, the embedding matrix was created with random entries. During the course of the training, the embedding layer was updated to learn similarities between the samples.[73]

As for the previous energy conditionings, all hyperparameters were kept the same. Since we have ten class labels, N_e was set to 10 and the size of embedding layer E was sampled as 50, 100, 206 and 512.

The 2D-DCWGAN energy embedding simulations were performed with the GAP and basin hopping datasets in the single GPU, multi GPU DP and multi GPU DDP code version.

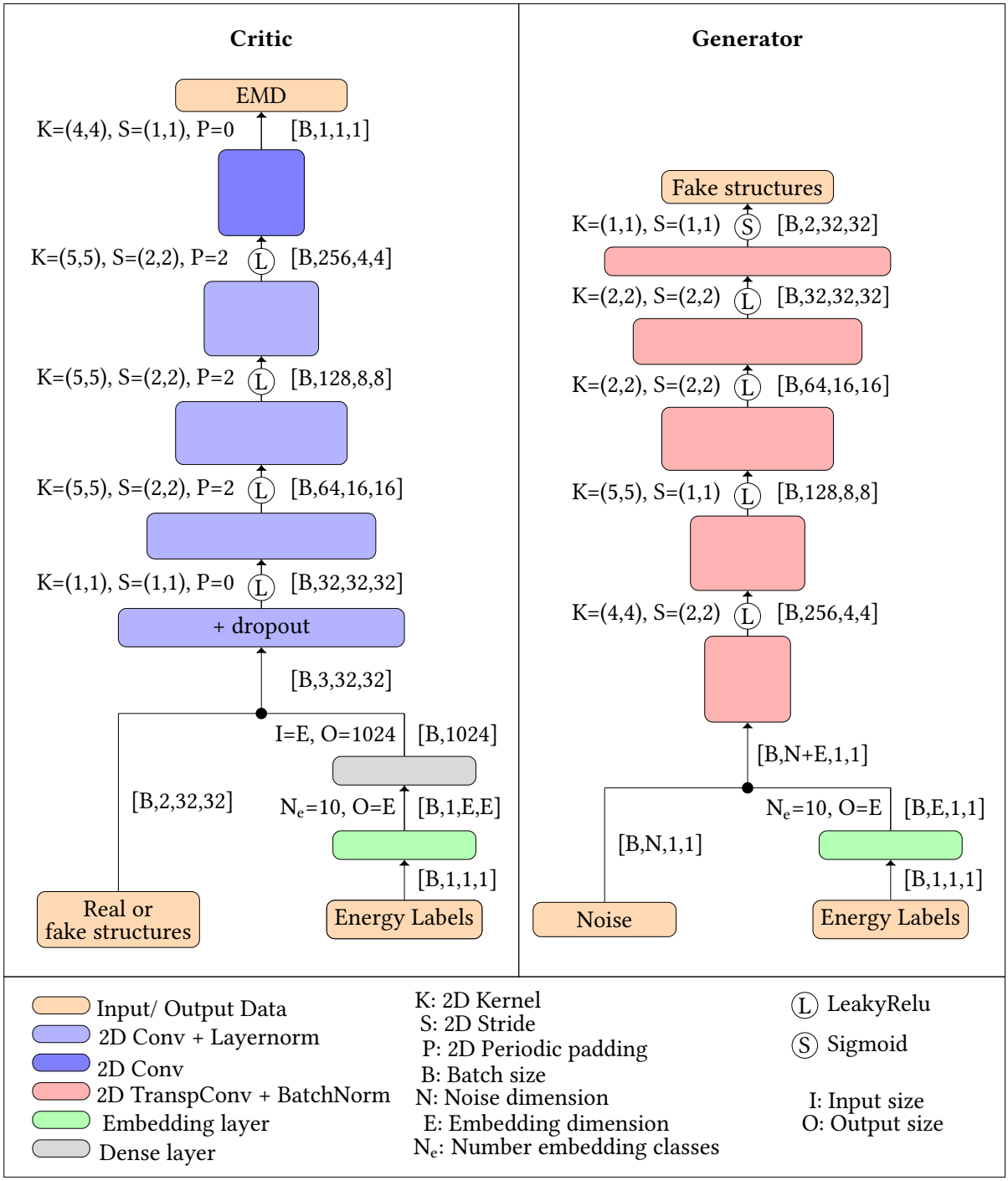


Figure 3.14: Computational architecture and computational details of the implementation of the 2D-DCWGAN with energy embedding.

3.3.3 Lattice regression in 2D-DCWGAN

In a similar fashion to [20], we added two auxiliary networks in our DCWGAN framework to predict the corresponding lattice lengths of our density-based structure inputs. The WGAN basically learns a conditional probability distribution $P(X|L)$ for the relationship between the chemical structures X and the lattice lengths L. The interested reader is referred to [72], where the mathematical background on the mutual information maximization is derived. Here, INFOGAN learns a conditional probability distribution by introducing an additional loss term for two auxiliary networks.

The computational architectures for the critic and the generator are illustrated in Fig. 3.16 and 3.17, respectively. To train the auxiliary networks, a new lattice loss function was introduced, analogously to [20]. The framework of [20] was implemented in Tensorflow Keras Version 1 utilizing the *tf.nn.sigmoid_cross_entropy_with_logits* loss function. We used the PyTorch analogon, the *MultiLabelSoftMarginLoss* (see Eq. (2.9)). For this purpose, the logits of the lattice output were also stored before the sigmoid layer was applied to them. The real lattice lengths $L_{\mathrm{real},i}$ were rescaled to $[0, 1]$ according to the minimum value $L_{\mathrm{real,min},i}$ and maximum value $L_{\mathrm{real,max},i}$ of the respective lattice orientation $i = x, y, z$:

$$L_{\mathrm{real,scaled},i} = \frac{L_{\mathrm{real},i} - L_{\mathrm{real,min},i}}{L_{\mathrm{real,max},i} \cdot 1.1 - L_{\mathrm{real,min},i}} \quad . \tag{3.1}$$

The flowchart in Fig. 3.15 shows the new training routine. Similar to the *Vanilla* 2D-DCWGAN, the real and fake structures were provided as an input for the critic, and for both the generator and critic the EMD-based WGAN loss is computed. As before, the gradient penalty gp was included in the critic loss. Again, the critic was updated five times in the inner loop, before the generator training begins in the outer loop. In both loops, an additional lattice loss was computed before the respective optimizer updated the critic or the generator. As seen, both the lattice lengths and logits were created on-the-fly during a forward pass through a neural network and can be easily integrated in the training routine.

As depicted in Fig. 3.16, the critic infers a lattice length for each structure in the inner training loop. The inferred lattice lengths of the real structures were then compared to the real lattice lengths. The MLSM loss was computed with the inferred lattice logits L_{reallog} and the real lattice lengths L_{real}

$$\begin{aligned} \mathrm{MLSM}(L_{\mathrm{reallog}}, L_{\mathrm{real}}) = -\sum_i L_{\mathrm{real},i} \cdot \log\left(\frac{1}{1+\exp(-L_{\mathrm{reallog},i})}\right) \\ + (1 - L_{\mathrm{real},i}) \cdot \log\left(\frac{\exp(-L_{\mathrm{reallog},i})}{1+\exp(-L_{\mathrm{reallog},i})}\right) \quad , \end{aligned} \tag{3.2}$$

and is added on top of the critic loss

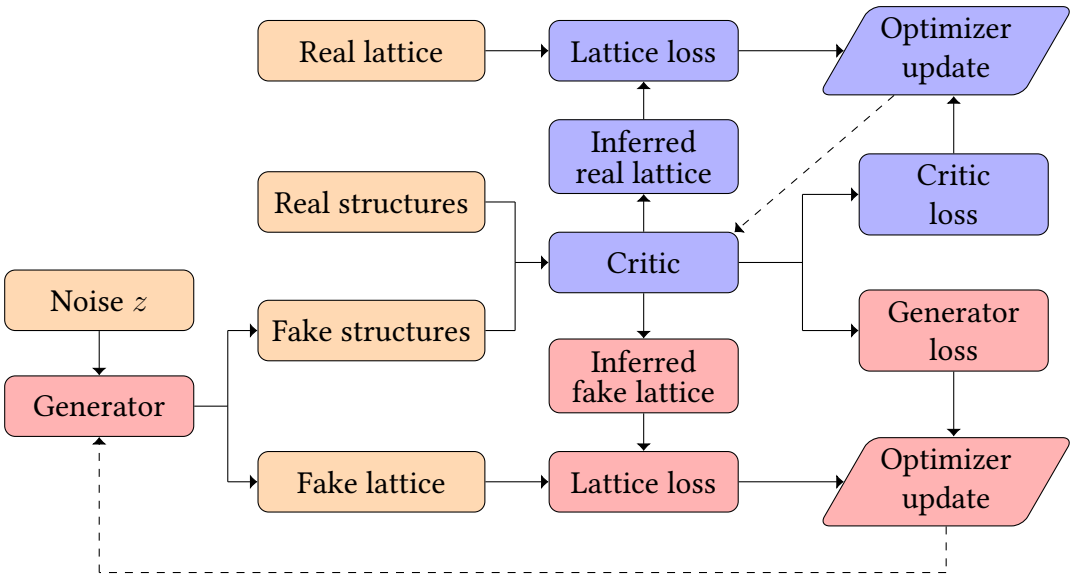


Figure 3.15: Code scheme for the encoding of the lattice lengths during the training process in a 2D-DCWGAN.

$$\begin{aligned} \mathrm{L}_{\mathrm{WGANlat,critic}} = \min_{g_\theta} \max_{d_\phi} & E_{z\sim P_{X'}}\left[d_\phi(g_\theta(z))\right] - E_{x\sim P_X}\left[d_\phi(x)\right] \\ & + \lambda_{\mathrm{gp}} \cdot E_{x_i\sim P_{gp}}\left[(\|\nabla_{x_i} d_\theta(g_\theta(z))\|_2 - 1)^2\right] \\ & + \mathrm{MLSM}(L_{\mathrm{reallog}}, L_{\mathrm{real}}) \quad . \end{aligned} \tag{3.3}$$

In two dimensions, the lattice losses for either the x,z or y,z lattice lengths can be computed simultaneously. After backpropagation, the gradients and the combined losses $\mathrm{L}_{\mathrm{WGANlat,critic}}$ were passed to the ADAM optimizer for the parameter update.

In the outer training loop, the generator was trained in a similar fashion. The critic infers a lattice length for the fake lattice lengths and as seen in Fig. 3.17 while the generator creates a fake lattice length concomitantly to the fake structures. The logit of the fake lattice length L_{fakelog} and the inferred fake lattice length $L_{\mathrm{inf,fake}}$ were inserted into the MLSM:

$$\begin{aligned} \mathrm{MLSM}(L_{\mathrm{fakelog}}, L_{\mathrm{inf,fake}}) = & -\sum_i L_{\mathrm{inf,fake},i} \cdot \log\left(\frac{1}{1+\exp(-L_{\mathrm{fakelog},i})}\right) \\ & + (1 - L_{\mathrm{inf,fake},i}) \cdot \log\left(\frac{\exp(-L_{\mathrm{fakelog},i})}{1+\exp(-L_{\mathrm{fakelog},i})}\right) \quad , \end{aligned} \tag{3.4}$$

and added on top of the generator loss

$$\mathrm{L}_{\mathrm{WGANlat,gen}} = \min_{g_\theta} \max_{d_\phi} E_{z\sim P_{X'}}\left[d_\phi(g_\theta(z))\right] + \gamma \cdot \mathrm{MLSM}(L_{\mathrm{fakelog}}, L_{\mathrm{inf,fake}}) \tag{3.5}$$

with hyperparameter $\gamma = 0.1$ for all simulations.[20] The backpropagated gradients and the combined $L_{WGANlat,gen}$ loss were handed to the ADAM optimizer which then updates the generator in the last step of one training epoch.

The computational details of the critic and the generator are listed in Fig. 3.16 and 3.17, respectively. The hyperparameters were kept the same as in the previous simulations.

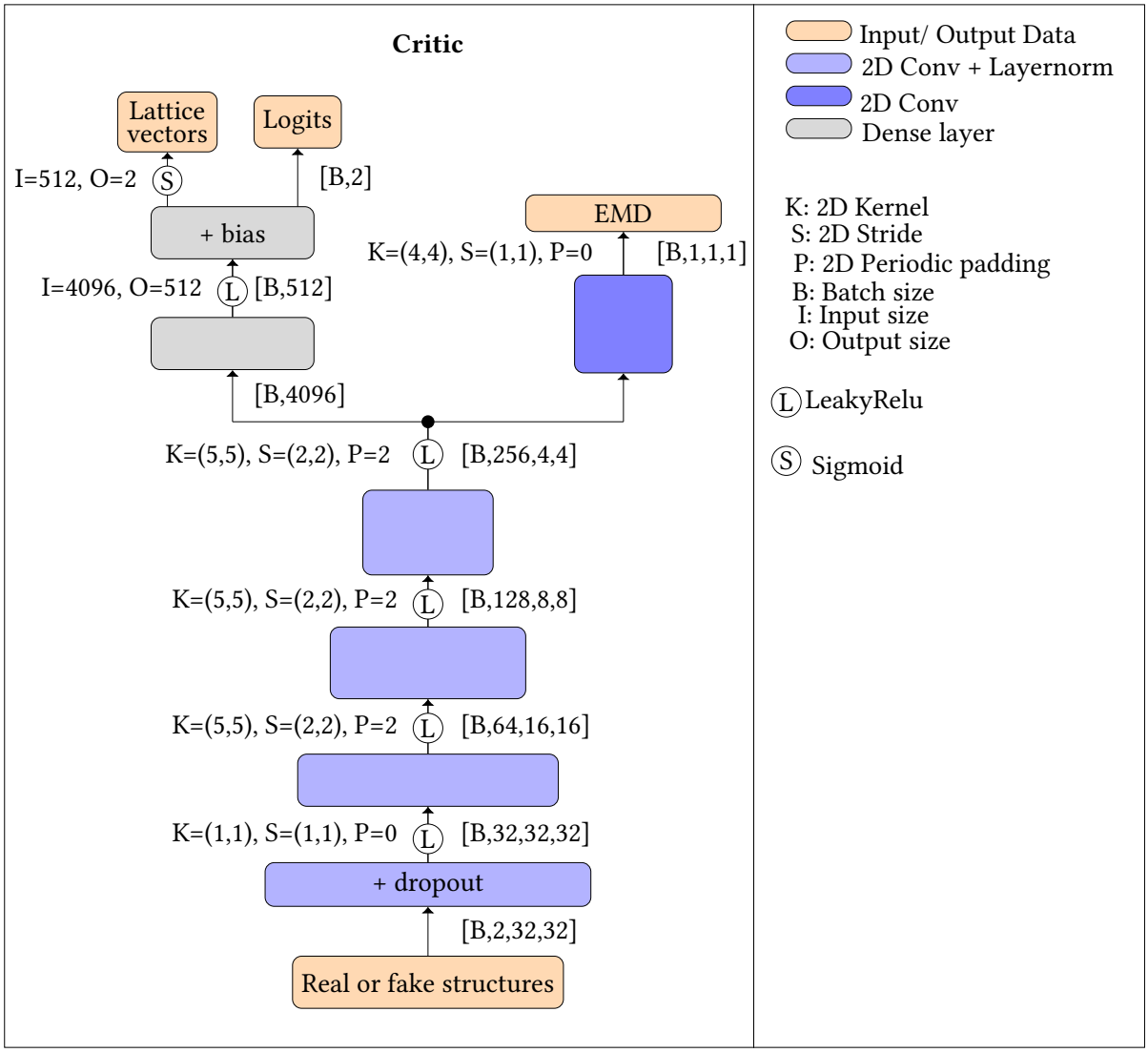


Figure 3.16: Computational architecture and computational details of the implementation of the critic in the 2D-DCWGAN with lattice prediction.

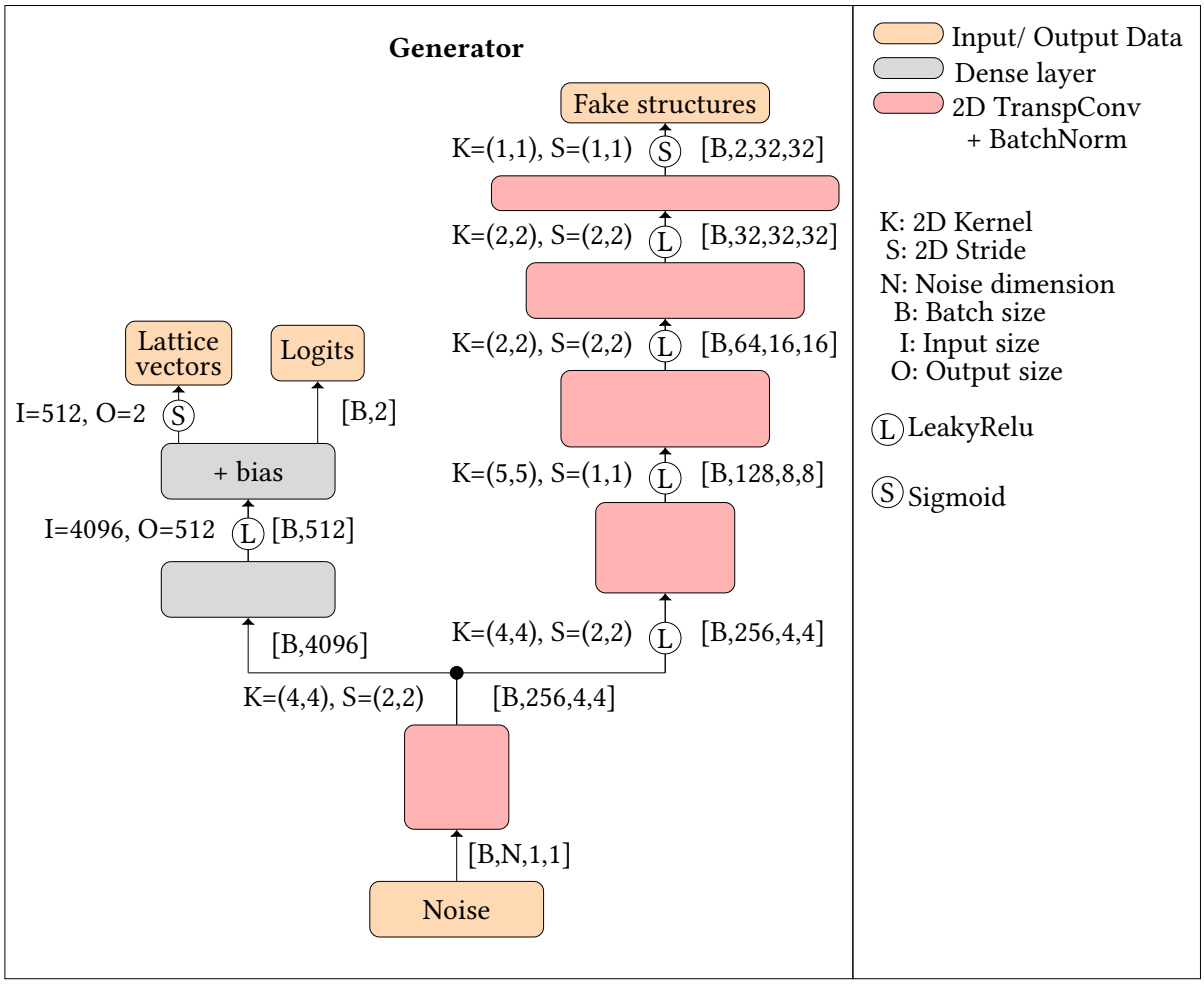


Figure 3.17: Computational architecture and computational details of the implementation of the generator in the 2D-DCWGAN with lattice prediction.

3.3.4 *Vanilla* 3D-DCWGAN

The aim of this work is to predict realistic three-dimensional structures for the RuO_2 catalyst system. Thus, in this final step, the unsliced, three-dimensional density-based input was imposed with three-dimensional convolutional and transposed convolutional layers. The same code routine as for the *Vanilla* 2D-DCWGAN (see Fig. 3.10) is executed. The computational architecture of the *Vanilla* 3D-DCWGAN is illustrated in Fig. 3.18. Generator and critic learning rates were both set to 0.001, the noise dimension was fixed to 512 and a maximum batchsize of 32 was set due to the limited space of 40 GB on the GPUs. The remaining hyperparmaters were adopted from above. The three-dimensional WGAN code was only implemented as a multi GPU DDP version with the basin hopping dataset.

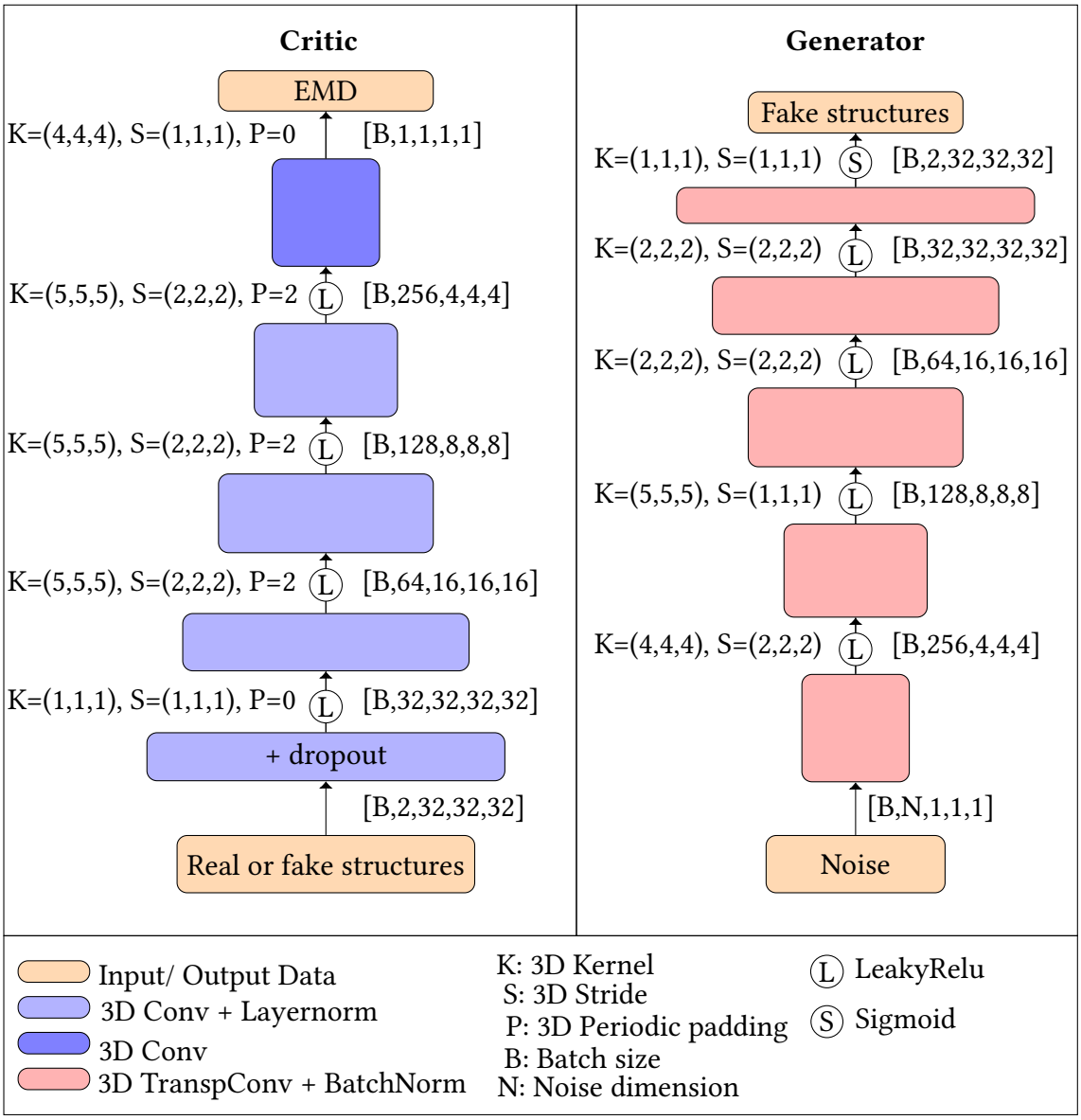


Figure 3.18: Computational architecture and computational details of the implementation of the *Vanilla* 3D-DCWGAN.

4 Results and Discussion

In this section, we will present the generated structures of the training for the in Section 3 listed computational architectures and discuss these results with respect to the quality of the density-based geometric output, the convergence of the loss functions, and explore how well different additional network features succeed in encoding structural energies or lattice lengths in our WGAN framework.

4.1 2D-DCWGAN

The initial model is a WGAN that takes two-dimensional density-based images as input. In the following, the results for the *Vanilla* 2D-DCWGAN are discussed, followed by the three different approaches to encode the energy - integer-based conditioning vectors, one hot encoded vectors and embedding layers - and by the lattice regression auxiliary network.

4.1.1 *Vanilla* 2D-DCWGAN

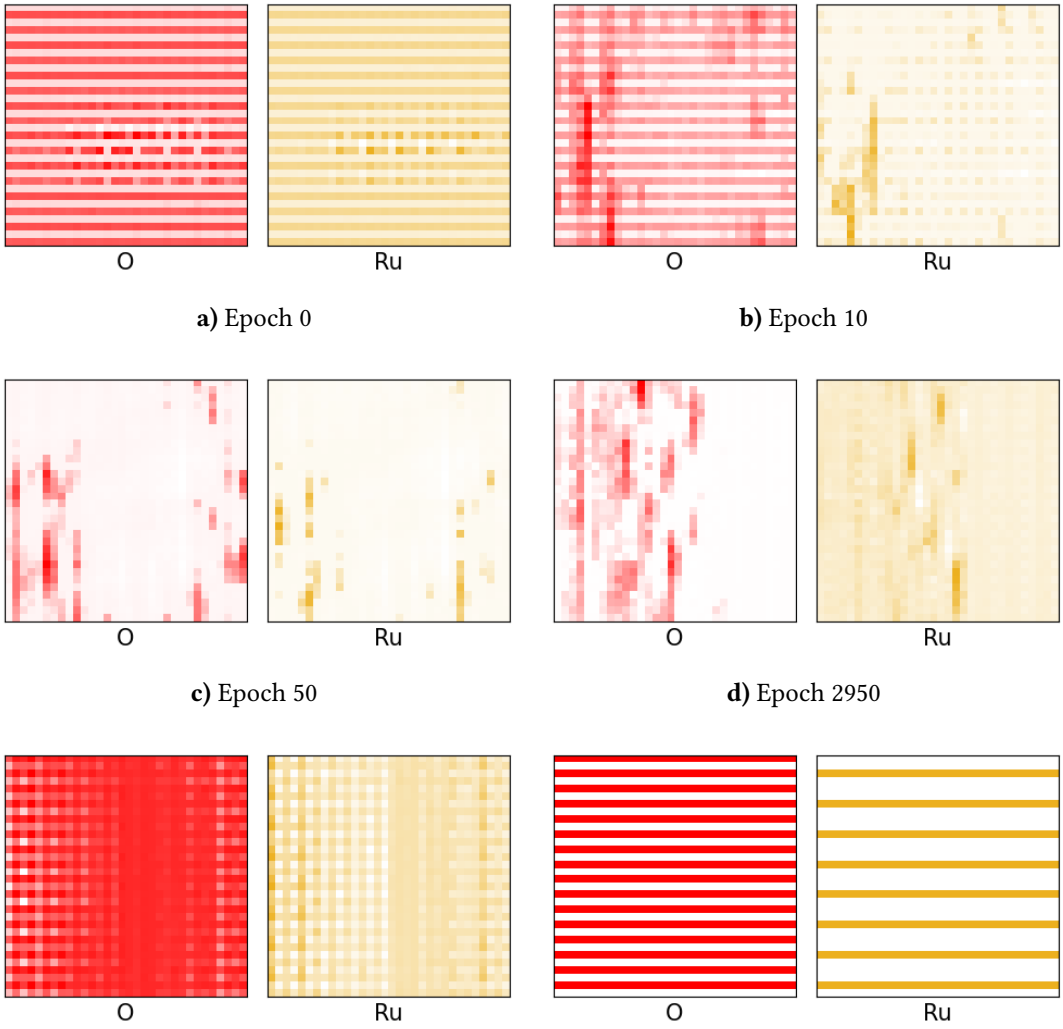
In this section, the learning process for the *Vanilla* 2D-DCWGAN is discussed with respect to the dataset designs from Section 3.1. The computational architecture from Section 3.3.1 and the concomitant hyperparameters are used for all following simulations.

Single-GPU Code with the non-extended GAP dataset

In this first network implementation, the non-extended GAP dataset with 148 structures was inserted as density-based input. The 148 structures were sliced 16 times in xz and yz direction, as described in Section 3.1. For both, the generator and critic a learning rate of 0.001 was set. The batchsize was 32.

In Fig. 4.1, the snapshots for one generated training structure in Epoch 0, 10, 50, 2950, and 3250 are displayed. In the beginning, the network learns to output horizontal stripes of alternating intensity in the background. As the training continues, the generator start to map pixels in closer proximity to each other such that the overall image starts to resemble positions of atom densities. However, compared to the images used as input shown in Fig. 3.7, the pixels are still quite uncorrelated and do not coincide with the sharp Gaussians in Fig. 3.7 that can be clearly separated from each other. In the generator state from Epoch 50 to 2950, the learning process of the mapping from the latent space was thus still in progress and the Gaussian densities started to become sharper more distinct from the background. In spite of the continued learning process, the image quality started to deteriorate at around Epoch 3200. From Epoch 3260 until the final training epoch 4790, the generator outputs only the

same horizontal stripe image. This is a clear indicator for the in Section 2.2.4 introduced phenomenon of a mode collapse, as the generator dropped all the other mode of the data distribution and outputs only the feature of stripes.



a) Epoch 0 **b)** Epoch 10

c) Epoch 50 **d)** Epoch 2950

e) Epoch 3250 **f)** Epochs 3260 - 4790

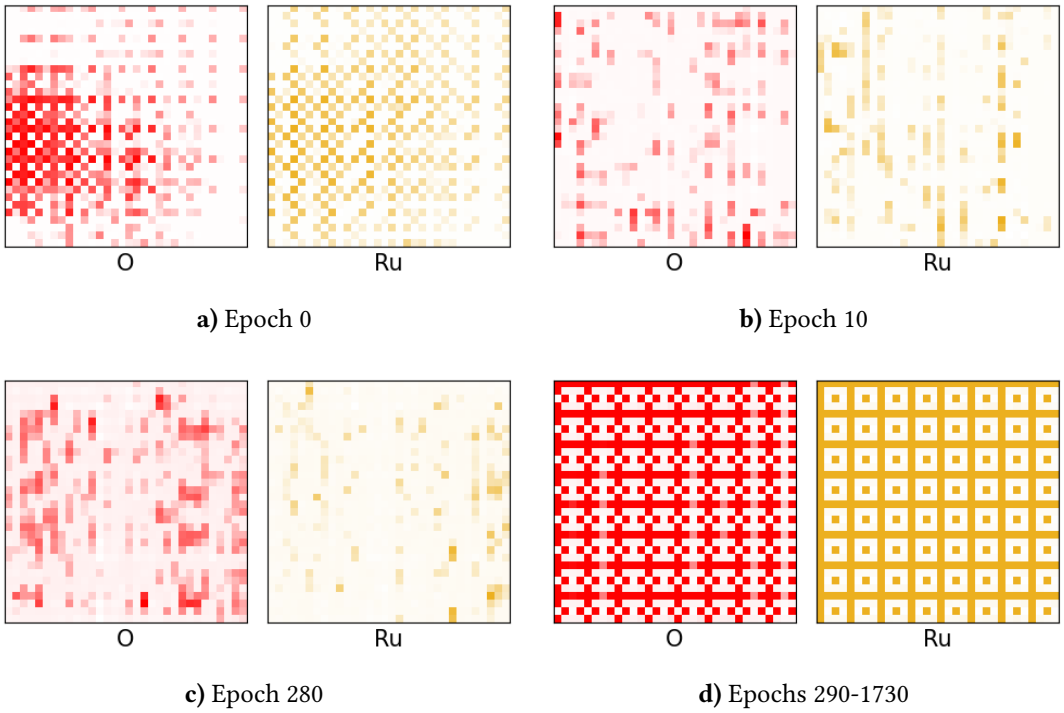
Figure 4.1: Generated structures from the training process of the *Vanilla* 2D-DCWGAN with the non-extended 148 structure GAP dataset at different epochs.

Single-GPU Code with the extended GAP datasets

As seen above, the training of the non-extended GAP dataset led to a mode collapse. Even varying the hyperparameters for the simulation, such as the learning rates or the optimizer parameters did not change the inevitable outcome of a mode collapse. To overcome this obstacle in the training process, we increased amount of atoms in each cell that was mapped to our 32x32 grid, as described in Section 3.1.

The results of the extended GAP datasets in the training process are illustrated in Fig. 4.2 for 148 structures and in Fig. 4.3. For these simulations, a batchsize of 32 and a learning rate of 0.001 was used.

For the training with the GAP dataset based on 148 structures, improvement was already seen in Epoch 0, as the generated structure does not contain any stripes. In this generated image, the clear distinction between the grouped pixels and the white, empty background shows similar to the real density-based input. From Epoch 10 to 280, the pixels in both channels start to assemble more, but the output is still clearly not converged. After Epoch 290, however, the generator encounters the problem of the mode collapse again, as seen in Fig. 4.2.



a) Epoch 0 **b)** Epoch 10

c) Epoch 280 **d)** Epochs 290-1730

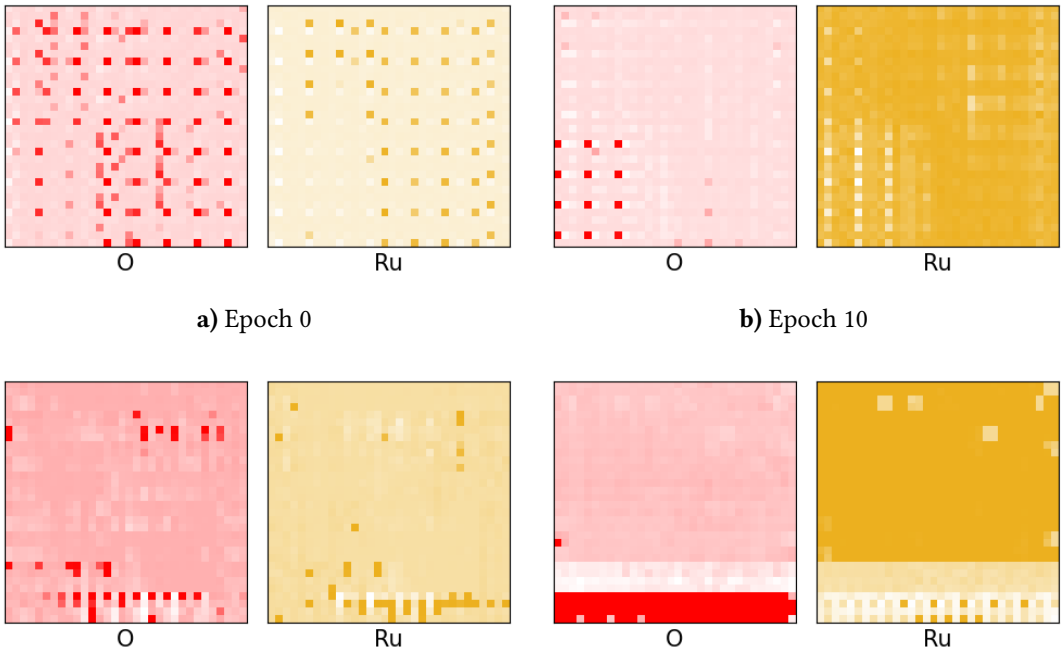
Figure 4.2: Generated structures from the training process of the *Vanilla* 2D-DCWGAN with extended GAP dataset based on 148 structures at different epochs.

For the training with the dataset based on 91 structures (see Section 3.1), the outcome is worse than for our original approach in Fig. 4.1. As seen in Fig. 4.3, the generator starts to produce single pixels of higher intensity in Epoch 0, they however do not assemble in groups to from Gaussian density shapes during the course of the training. Additionally, the background intensity is never close to zero. Thus, the desired property of a distinction between Gaussians representing atomic positions and a white background representing empty cell spaces is not covered in the generator. After Epoch 80, the generator is stuck in a mode collapse, as seen in Fig. 4.3.

From the two simulations with the small (91 structures) and big (148 structures) extended GAP dataset, we conclude that the increase of the amount of atoms leads to an improvement

of the dataset representation. The critic of the network can start to distinguish the geometric features of the periodically repeating atoms represented as Gaussian densities and the empty background better and can thus distinguish better the generator output from the real data. The generator vice versa has then to adapt the improved critic by improving the image quality of the generated structures.

It is evident that the output quality for the increased cells in the dataset with 91 structures has significantly decreased. We attribute this deterioration to the fact that the overall dataset size was reduced 18,944 to 11,648 2D images, as explained in Section 3.1. Since WGANs do not only depend on the quality of the database, but on the database size as well, we conclude that the idea to remove non-physical structures is an interesting approach to improve the accuracy of possibly generated structures, but if the dataset size is reduced too much, it hinders the data-driven learning process.



a) Epoch 0 **b)** Epoch 10

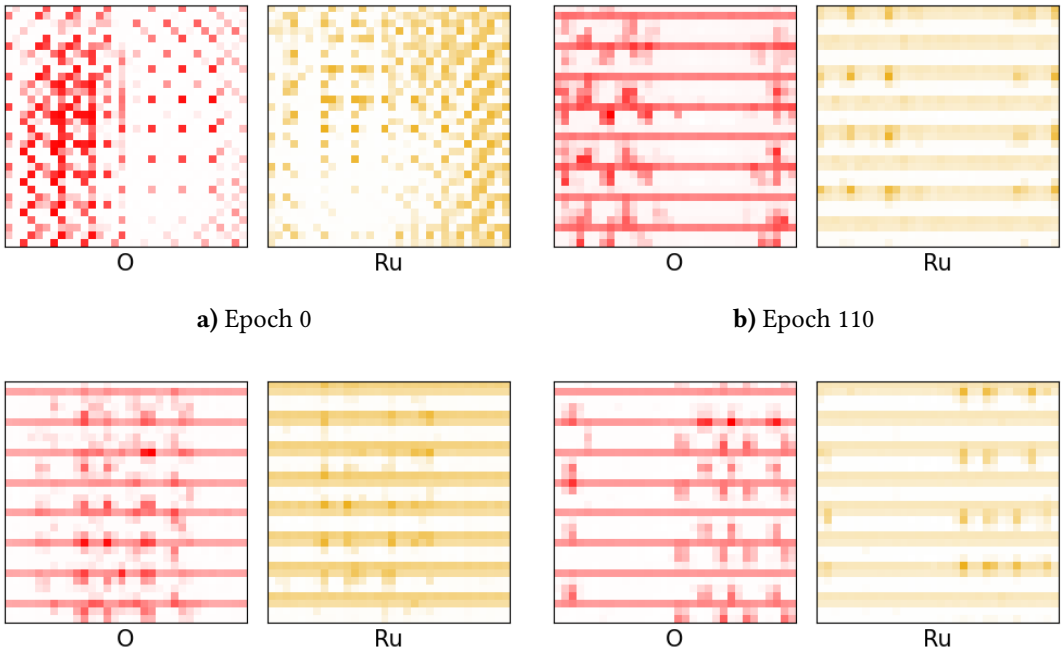
c) Epoch 60 **d)** Epochs 80-5570

Figure 4.3: Generated structures from the training process of the *Vanilla* 2D-DCWGAN with extended GAP dataset based on 91 structures at different epochs.

Multi-GPU Code with the basin hopping dataset

In this simulation, we tested the influence of the larger basin hopping dataset with 57,806 structures on the training success of our *Vanilla* 2D-DCWGAN. To have enough computational capacity to process this amount of data within the epochs, the code was upgraded to single-node multi-GPU computations DDP package[67] package, as described in Section 3.3 and run on four GPUs at the same time. The learning rates were set to 0.001 and two simulations of batchsize 512 and 1024 were investigated.

In Fig. 4.4, generated structures for batchsize 512 are illustrated. At Epoch 0 single pixels start to assemble. Starting from Epoch 110 to Epoch 2150, the characteristic Gaussian densities of the atomic position can be seen. However, the generated images display horizontal stripes in the background, similar to the ones from the mode collapse in Fig. 4.1. Different to Fig. 4.1, the stripes in Fig. 4.4 are less intense, smaller and the pixels representing atomic densities are still switching positions between the images.



a) Epoch 0 **b)** Epoch 110

c) Epoch 660 **d)** Epoch 2150

Figure 4.4: Generated structures from the multi-GPU training process of the *Vanilla* 2D-DCWGAN with basin hopping dataset and a batchsize of 512 at different epochs.

As described in Section 2.2.4, a mode collapse can in the worst case lead to a generator that only learns the representation of one single mode (see Eq. (2.27)), as in Fig. 4.1 but it can also refer to the situation in which the generator learns to represent a few modes of the multi-modal probability distribution. Since the atomic densities are still being placed at different positions throughout the training process, we can assume that the generator is currently in a mitigated form of a mode collapse.

This assumption is supported by an inspection of the generator loss and critic loss over the training process. As illustrated in Fig. 4.5, the generator and critic loss are in the same order of magnitude for the first hundred epochs. The generator loss even decreases in the beginning which means that the generator learned a new way to generator fake structures such that the critic is not capable of distinguish real from fake samples as good as before. Ideally, the loss curves of the generator and critic would show shifted oscillations indicating that one

of the networks learned a new feature and the other network has to adapt its parameters again. But after around Epoch 100, the generator loss continuously increases, whereas the critic loss goes to zero for the rest of the training process. This is an indicator that the critic is already trained well enough to perfectly distinguish between fake and real samples leading to vanishing gradients in the critic. If the gradients for the critic, however, start to vanish, it becomes impossible for the generator to improve.

Before the generator runs into a mitigated mode collapse at the end of the training process, the generator was capable to learn the shape of the Gaussian densities in the first hundred epochs of the training process, as indicated by the small oscillations in the loss function. Afterwards, it stagnates in a mitigated mode collapse which is verified through the monotonous increase of the generator loss and the vanished critic loss.

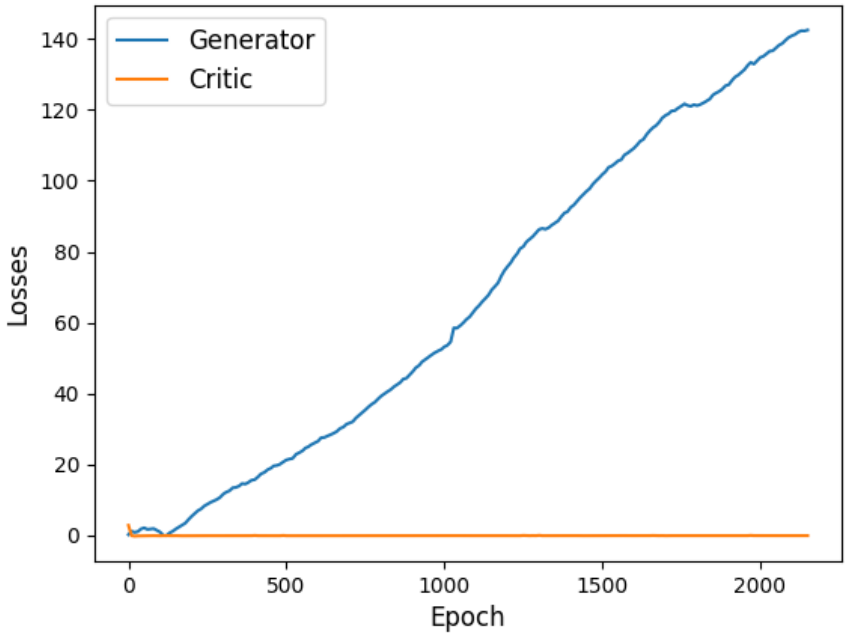
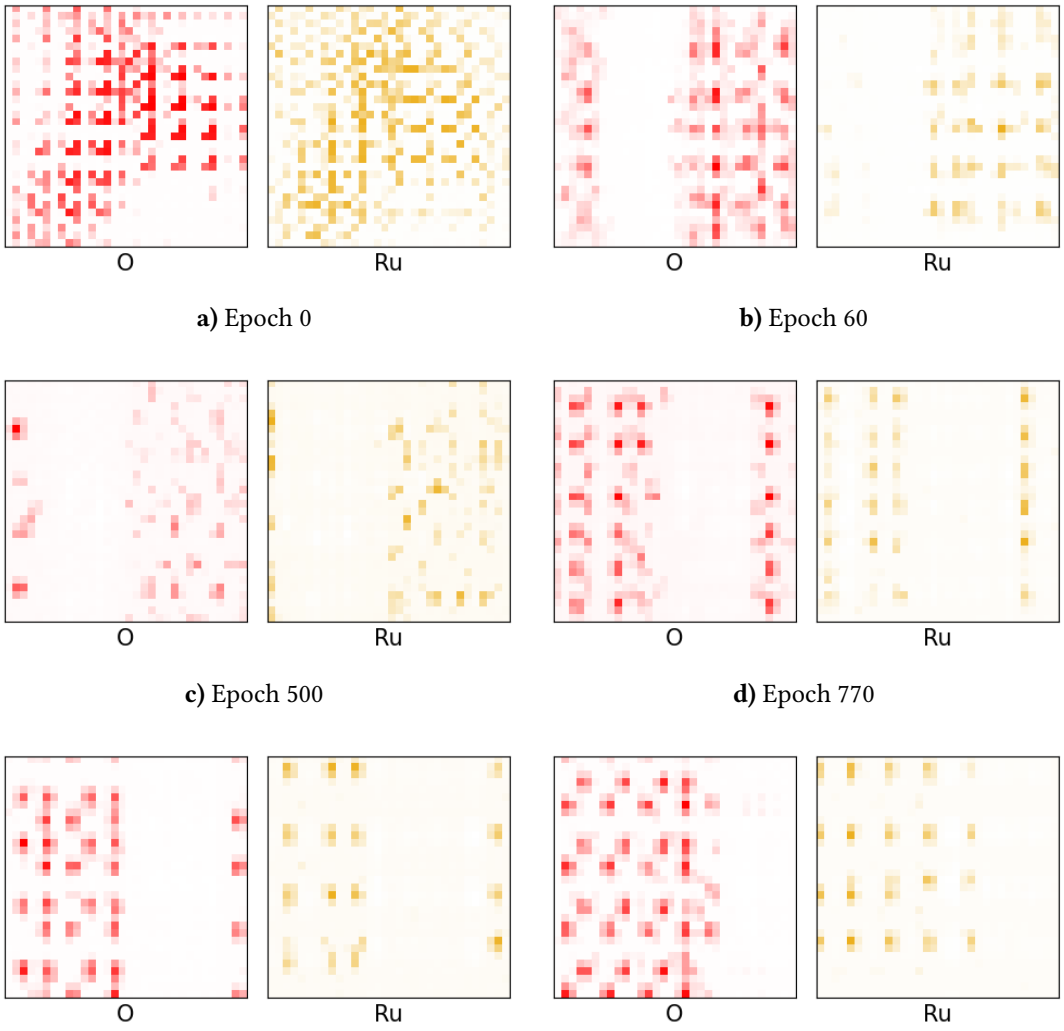


Figure 4.5: Losses during the training process of the *Vanilla* 2D-DCWGAN with basin hopping dataset and batchsize 512.

In Fig. 4.6, the training progress of the same simulation with a doubled batchsize - 1024 - is illustrated. For Epoch 0, the generated structure is relatively similar. Both atomic densities show the same cluster pattern. Interestingly, the generated images in Fig. 4.6 exhibit no horizontal stripes in contrast to Fig. 4.4.

To understand this, we will briefly discuss the influence of the batchsize on the training process. In each epoch, the batched data is passed through the WGAN framework. For both, the generator and the critic, the loss is computed for each sample in the batch and then averaged over the whole batch. If some samples strongly deviate, their impact on the overall loss can be

mitigated through a larger batchsize. This means that the accuracy falls for larger batchsizes.[74] The impact of an increased batchsize on the gradients of the SGD and ADAM optimizers is thoroughly explored in [74]. The authors argue that an increased batchsize has the same effect as an decreased learning rate on the training process. Both results in a reduction of the gradient noise on the loss surface, which allows the optimizer to converge faster in the global minimum without stranding in a local minimum.[74]



a) Epoch 0 **b)** Epoch 60

c) Epoch 500 **d)** Epoch 770

e) Epoch 1150 **f)** Epoch 2060

Figure 4.6: Generated structures from the multi-GPU training process of the *Vanilla* 2D-DCWGAN with basin hopping dataset and a batchsize of 1024 at different epochs.

Based on this argumentation, we deduce that the doubled batchsize allowed the optimizers to converge faster in the global minimum of our loss function. This conclusion is supported by the generator and critic loss diagram in Fig. 4.7. In the beginning, the critic loss decreases, as well as the generator loss. This is the initial phase where the first modes of the dataspace

are learned. Afterwards, the generator loss increases again and reaches a maximum at around Epoch 500. One generated structure at Epoch 500 is shown in Fig. 4.6. It is clearly visible that image quality deteriorated in comparison to previous or later generated structures in Fig. 4.6. The assembled pixels have a low intensity and the pixels are grouped in a non-systematic way. After Epoch 500, the generator loss starts to decrease again and already at Epoch 700, the image quality of the atomic density has significantly improved again, as illustrated in Fig. 4.6. For Epochs 1000 to 2000, the generator loss consistently oscillates around values of -1 and the critic loss is close to zero but exhibits small oscillations around 0. Based on the converged loss curves in Fig. 4.7 and the high sample quality in Fig. 4.6, we assume that this training process was successful.

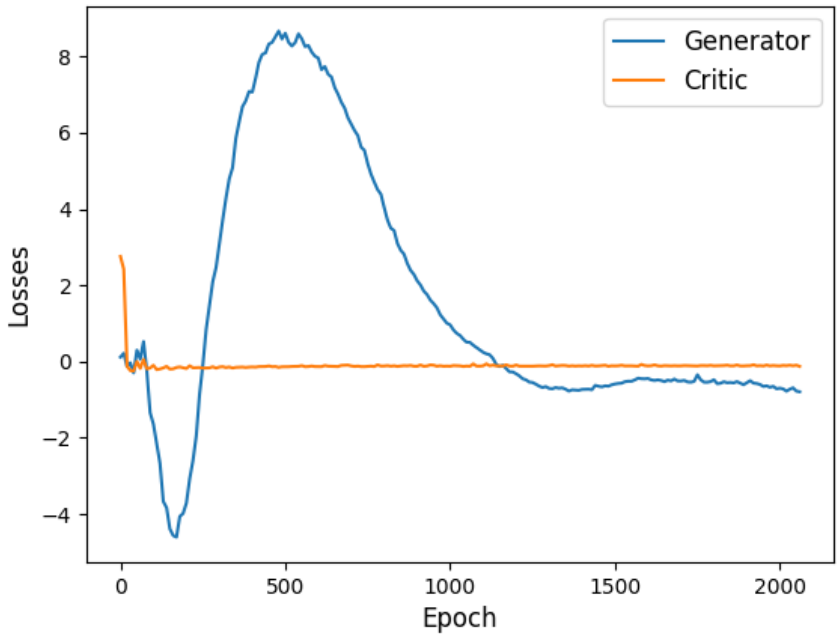


Figure 4.7: Losses during the training process of the *Vanilla* 2D-DCWGAN with basin hopping dataset and batchsize 1024.

After a successful training process, one of the key questions in machine learning arises: Is our model overfitted?[75–77] Specifically to our GAN framework, this question entails if the generator parameters θ are sufficiently tuned such that $P_{X'} \approx P_X$ holds and that $g(z)$ creates a new sample? Or is the generator only capable to memorize our input datasets and is not capable of producing new samples $g(z)$ that are not part of the finite training set? The latter situation corresponds to an overfitted model.[78]

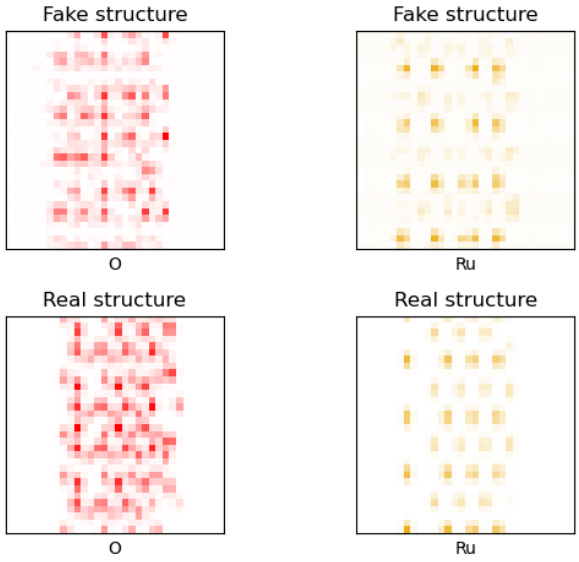
Hence, to check whether our model is overfitted, we compare each saved generated training structure against the 57,806 structures from the basin hopping dataset. As a quantitative measure of the similarity of the images, we use the *skiimage*-implementation of the Structural Similarity Index Measure[79] (SSIM). SSIM explores the dependencies of spatially proximate pixels in an image introducing three combined comparison functions for the contrast, the

structure and the correlation of the pixels. For more details, the interested reader is referred to [79]. We chose a window size of 5 for the SSIM, similar to the voxel size for our Gaussian densities.

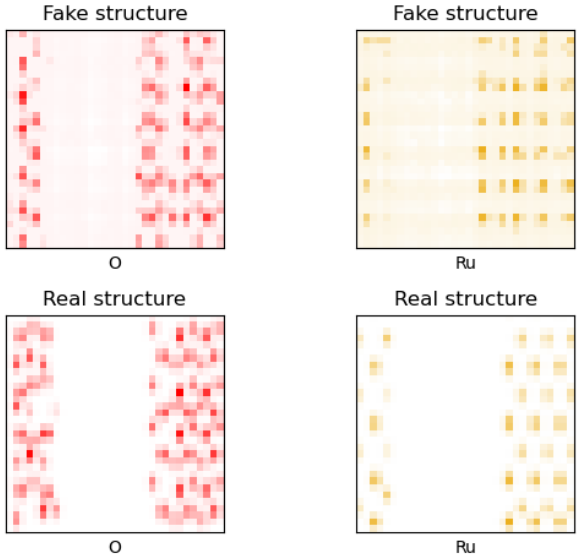
The SSIM was computed for the O and Ru channels, separately. The image indexes from our database that had the most similar SSIM values in comparison to the training data were saved. From 205 analyzed training structures, SSIM identified 24 similar images from the database and their comparability was confirmed by visual inspection. In Fig. 4.8, two of these 24 found structures are illustrated next to the corresponding structure from the dataset.

Comparing the fake structures and the real structures in Fig. 4.8, they show similar features such as similar arrangements and intensity of the pixels and the same positions for the empty background, aka the vacuum space of our cell. However, the images can clearly be distinguished because the positions of the main intensities alter especially for the Ru atom densities and are thus corresponding to different structures. We can confidently conclude that the generator did not memorize the dataset. This finding is in agreement with the literature since the generator never explicitly sees the real structures, it only improves through the critic's feedback.[78]

But we can use this quantitative similarity measure to prove that some real structures are similar to the fake structures which is a further indicator for a successful training process of the computational architecture of our *Vanilla* 2D-DCWGAN. As we now successfully tuned the hyperparameters for the *Vanilla* 2D-DCWGAN and proved that the present computational architecture is capable to generate new structures that still resemble the original dataset, we will proceed to explore the three different possibilities - conditioning vectors, one hot vectors and energy embedding - to introduce energy class labels in our WGAN framework. The results shall be discussed with respect to the effects of the latent space encoding on the generator output.



a) Epoch 0



b) Epoch 110

Figure 4.8: Comparison of generated training structures (fake structures) with dataset structures (real structures) to check for overfitting in the multi-GPU training process of the *Vanilla* 2D-DCWGAN with basin hopping dataset and batchsize 1024. The real structures were assigned to the fake structures based on their SSIM[79] value.

4.1.2 Integer-Based Conditioning Vectors

As described in Section 3.3.2, the integer-based conditioning vectors of size C are constructed by a repetition of the corresponding class label (0 to 9). The integer-based conditioning vectors are either concatenated to the noise z for the generator or to the fake or real structures for the critic. The computational architecture of the conditioning 2D-DCWGAN is illustrated in

Fig. 3.13.

Multi-GPU Code with the extended GAP datasets

Three simulations with conditioning length 100, 206 and 512 were started for the two extended GAP datasets consisting of 91 and 148 original structures. The simulations for the conditioning lengths 100 and 512 ended after the first hundred epochs in mode collapses, whereas conditioning length 206 did not collapse. We thus want to discuss the quality of the density-based images with respect to the integer-based conditioning vectors of length 206.

Before we can evaluate the effect of the latent space encoding, we should first analyze which features we actually expect. In Fig. 4.9, two image slices from the same structure with energy class label 7 are shown. The slices exhibit inherently different features, with altering number of atoms and positions. For any two slices, the cell width can vary as well. Nevertheless, both slices originate from the same structure. The energy of a three-dimensional structure is determined by its three-dimensional nuclear coordinates. As these structures are then sliced and presented as separate two-dimensional inputs with the same energy class label to the neural networks, the whole structure-energy relationship is distorted. It is thus only natural that our 2D representation will not be capable to connect the class labels with structures of higher or lower energy. Instead, the WGAN could start to connect random geometric features, like the position of the cell block and the vacuum layer to the class label or it could show no connection at all.

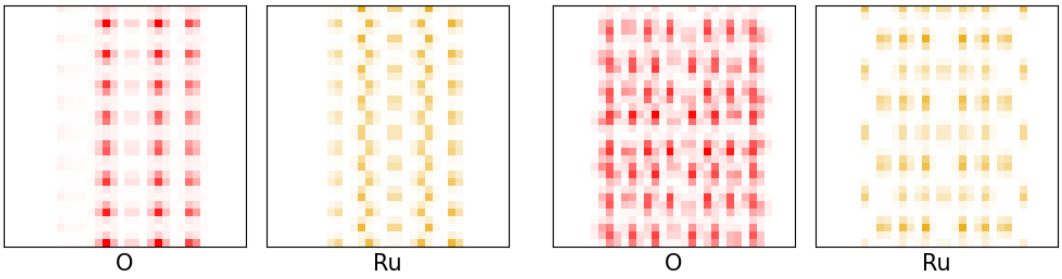
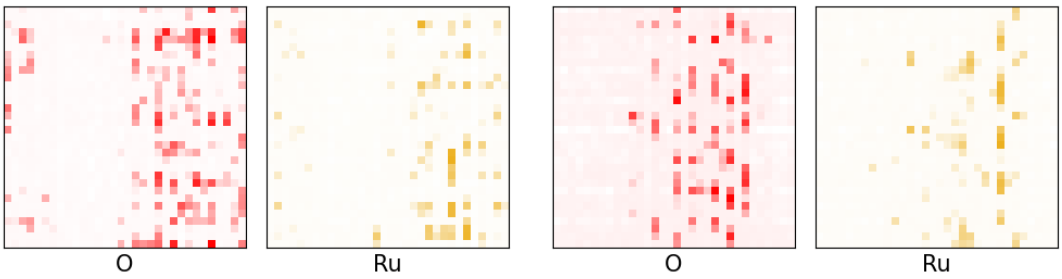


Figure 4.9: Two 2D slices from one structure of the 148 GAP dataset with energy class label 7.

Coming to the simulation for conditioning length 206, generated structures from the training process of an integer-based conditioning 2D-DCWGAN with the 148 structures GAP dataset for different epochs are illustrated in Fig. 4.10. During the process of the training, no mode collapse occurred. In these images, there is a clear distinction between pixels of low intensity that form the empty background and pixels of higher intensity that shall represent atomic positions. However, the pixels do not form assembled atomic densities yet. In the course of the training, the pixels of higher intensity start to group closer together and the images start to exhibit a cell area and a vacuum area. The training process is thus not converged yet but as elaborated in Section 3.3, the GPU memory allocation of the DP package leads to a poor

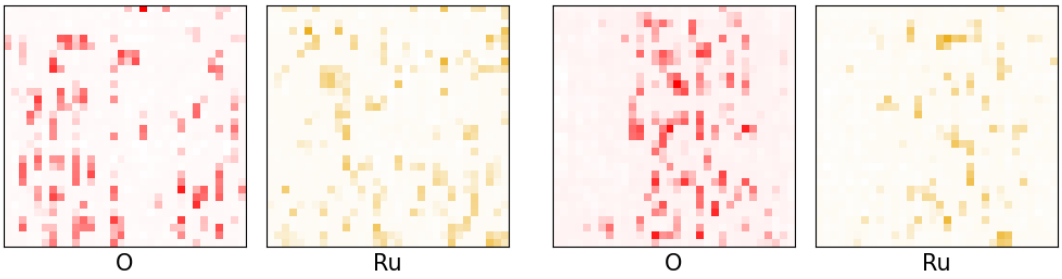
performance and an optimization of this simulation is not pursued.



a) Epoch 70 **b)** Epoch 450

Figure 4.10: Generated structures from the multi-GPU training process of the integer-based conditioning 2D-DCWGAN with extended 148 structure GAP dataset, batchsize 32 and conditioning vector size 206.

In Fig. 4.11, generated structures from the training process of an integer-based conditioning 2D-DCWGAN with the 91 structures GAP dataset and conditioning length 206 for different epochs are illustrated. Surprisingly, compared to the previous *Vanilla* 2D-DCWGAN, the simulation is not collapsing. We assume that due to the appended conditioning vector, the representation size of our dataset increased and concomitantly stabilized the training process. The behavior of the learning process is similar to the one from the conditioning 2D-DCWGAN trained with the 148 extended GAP dataset (see Fig. 4.10).



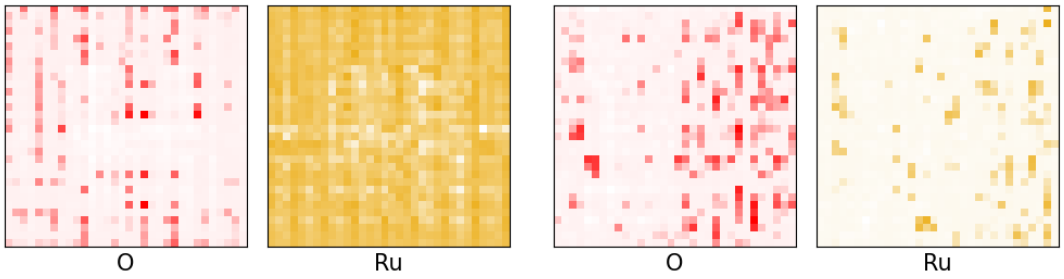
a) Epoch 70 **b)** Epoch 740

Figure 4.11: Generated structures from the multi-GPU training process of the integer-based conditioning 2D-DCWGAN with extended 91 structure GAP dataset, batchsize 32 and conditioning vector size 206.

As the simulations for conditioning lengths 100 and 512 stranded already after the first hundred epochs in a mode collapse and the conditioning length 206 is not collapsing, we assume that a ratio of 1:2 between integer-based conditioning and dimension for our latent space is beneficial for the training process. With regard to the non-converged outcomes, we have to be careful with further assumptions but we can already conclude that the conditioning length

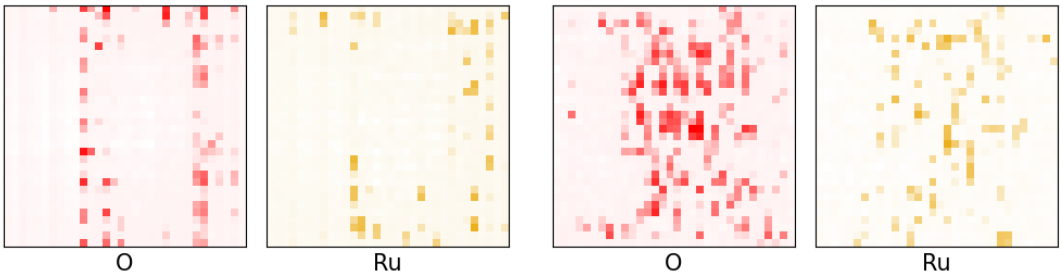
C is an important hyperparameter.

To examine the effect of our latent space design with integer-based conditioning vectors, we investigate how different labels influence the output of the generator. Fig. 4.12 shows two generated structures, one for label 0 and one for label 9 from the generator that was trained on the 148 structure GAP database until Epoch 450. Both images show uncorrelated noise and are two representative examples for the appearance of the majority of randomly generated fake structures with this generator. Thus, the training process is presumably still in its early stages. Hence, the latent space encoding is not a feature that the generator was capable of learning quickly during the training process. In Fig. 4.13, similar images are created by the generator that was trained on the 91 structures dataset until Epoch 740. For this simulation, we find that for the labels 0 and 9, characteristic geometric feature start to emerge. The high intensity pixels start to be more located in the middle of the cell for label 9, whereas for label 0, the pixels rather start to form bands. Both forms are represented by the two examples in Fig. 4.13. Due to the smaller dataset, the training process run almost twice as many epochs in the same time as in the previous simulation. This means that the generator had the double amount of training time and can thus start to connect geometric features in the picture with the latent space encoding. However, as these computations are not converged, we cannot make further deductions.



a) Label 0 **b)** Label 9

Figure 4.12: Generated structures produced with fixed labels by the until Epoch 450 trained generator of the integer-based conditioning 2D-DCWGAN with extended 148 structure GAP dataset, batchsize 32 and conditioning size 206.



a) Label 0 **b)** Label 9

Figure 4.13: Generated structures produced with fixed labels by the until Epoch 740 trained generator of the integer-based conditioning 2D-DCWGAN with extended 91 structure GAP dataset, batchsize 32 and conditioning size 206.

The benchmarks we can thus deduct from these simulations refer to the stability of the training process with respect to the image quality of the geometric output, i.e. the O and Ru channels, and the question if the generator starts to react sensitive to the latent space design. When we proceed with energy conditioning vectors for the three-dimensional WGAN framework, these benchmarks can be used as starting points for the hyperparameter optimizations. The fact that the computations for both GAP datasets resulted in mode collapses for image-based conditioning vectors sizes 100 and 512 but not for 206 is a clear indicator, that the sweet spot for an integer-based conditioning length is located somewhere around 200.

We therefore interpret this simulation as a proof of concept for the latent space design that definitely needs to be more thoroughly investigated in a more efficient implementation of a three-dimensional framework with longer runtimes.

4.1.3 One Hot Encoding Vectors

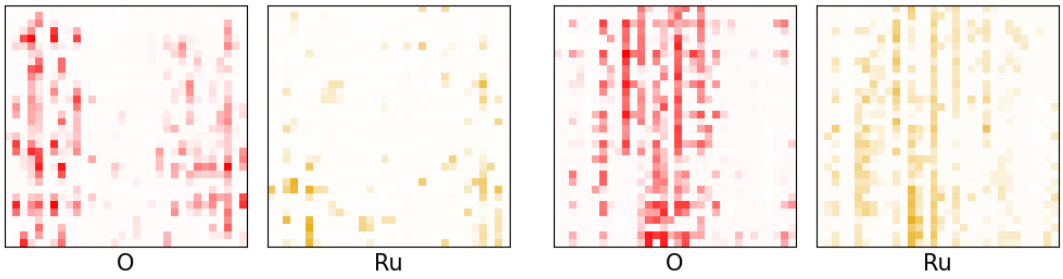
As introduced in Section 3.3.2, the one hot vectors encode the energy class labels in binary representations. Similar to the integer-based conditioning vector approach, the three-dimensional structures from the dataset are sliced and fed as separate two-dimensional inputs to the one hot encoded WGAN. This leads to a loss of correlation information between the structure input and the energy label. In the following, we will thus only examine the stability of the training process with respect to the image quality of the generated structures.

Multi-GPU Code with the extended GAP datasets

For the two extended GAP datasets consisting of 91 and 148 original structures, three simulations of one hot encoding length 100, 200 and 500 were started. The simulations for the one hot encoding lengths 100 and 500 ended in mode collapses. The computations for 200 digits long one hot encoded vectors did not collapse and will be discussed in the following.

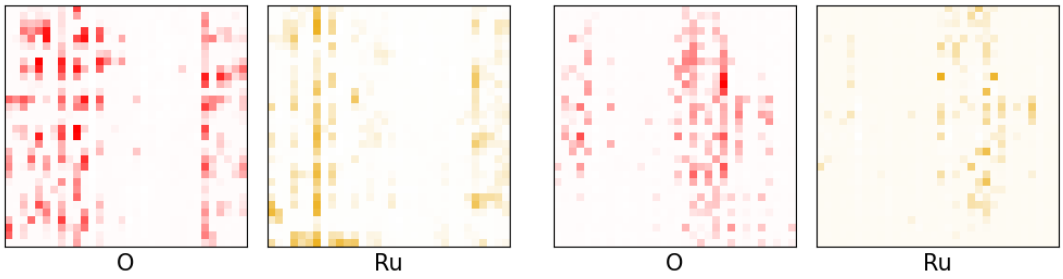
In Figs. 4.14 and 4.15, one generated structure from the beginning and one from the end of the multi-GPU training process are shown for one hot vector length 200. As in the previous simulation, the WGAN training process was stopped before convergence was reached and the pixels do not resemble Gaussian densities or distinct areas for the cell content and the vacuum layer. Analogously to before, for both trained generators, the output is investigated with respect to distinct class labels. Random geometric features and pixel shapes and intensities were found across all class labels in the generator output. Two examples for labels 0 and 9 for both trained generators are illustrated in Figs. 4.16 and 4.17.

We thus infer that in contrast to the conditioning vector representation, the generator was not yet capable to connect distinct geometric properties with the one hot encoded class labels. This raises the question if the one hot representation is not as suitable for our approach as the conditioning vectors. Furthermore, the training process for the one hot encoded vectors appears to be less stable. The mode collapse occurred significantly earlier for the one hot vector lengths 100 and 500 than for their integer-based conditioning counterparts. Additionally, the generated structures in both Figs. 4.15 and 4.17, show an overall decrease for the pixel intensity. This is true for all structures that were created with the until Epoch 730 trained generator and could be an indicator for an incoming mode collapse. The decreasing pixel intensity might be related to the inherent nature of the binary representation. For a one hot vector length of 200, 180 additional zeros were appended to the latent space and to the real or fake structures in the training loop. The values for the pixels intensities of the real structures are located on the interval $[0, 1]$. We thus assume that the increased amount of zeros can distort the training process of our density-based data.



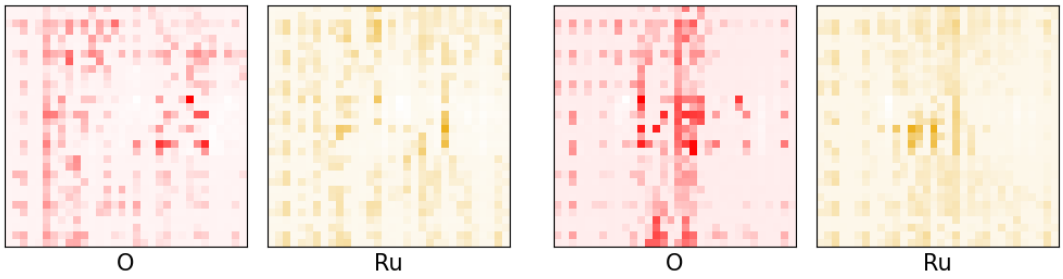
a) Epoch 60 **b)** Epoch 440

Figure 4.14: Generated structures from the multi-GPU training process of the one hot encoded 2D-DCWGAN with extended 148 structure GAP dataset, batchsize 32 and conditioning vector size 200.



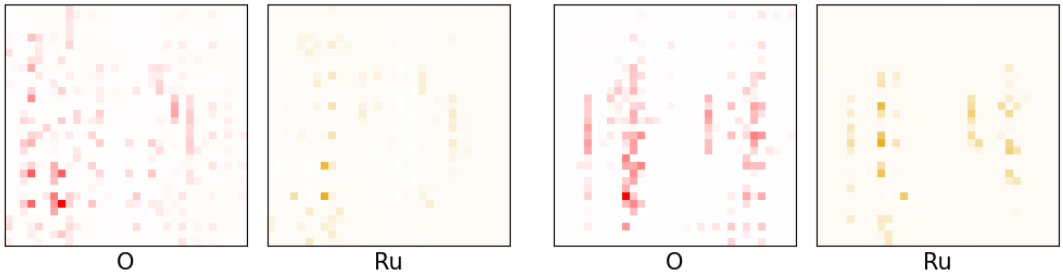
a) Epoch 60 **b)** Epoch 730

Figure 4.15: Generated structures from the multi-GPU training process of the one hot encoded 2D-DCWGAN with extended 91 structure GAP dataset, batchsize 32 and conditioning vector size 200.



a) Label 0 **b)** Label 9

Figure 4.16: Generated structures produced with fixed labels by the until Epoch 440 trained generator of the one hot encoded 2D-DCWGAN with extended 148 structure GAP dataset, batchsize 32 and conditioning size 200.



a) Label 0 **b)** Label 9

Figure 4.17: Generated structures produced with fixed labels by the until Epoch 730 trained generator of the one hot encoded 2D-DCWGAN with extended 91 structure GAP dataset, batchsize 32 and conditioning size 200.

Concluding, we believe that for the present dataset and investigation, the integer-based

conditioning vectors outperform one hot vectors as a representation for the energy class labels.

4.1.4 Energy Embedding

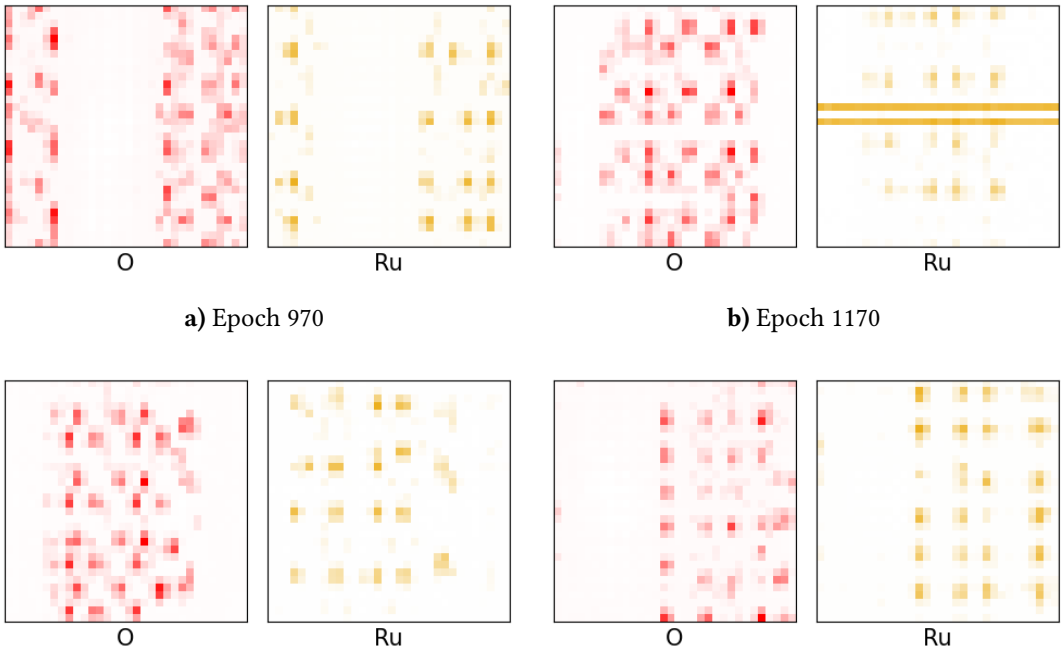
As described in Section 3.3.2, an additional network layer is augmented on the critic and generator networks to encode the energy class labels. Therefore, the energy class labels were inserted into the WGAN framework without further modification. In the following, we will analyse the image quality of the two atom channels with respect to the dimension of the embedding layer.

The first simulations were run based on the multi-GPU DP Code for both GAP datasets. Three different embedding layer sizes, i.e. 100, 200 and 500 for the sake of comparability to the other two simulations were started. Unsuccessfully, all three embedding layer sizes led to a mode collapse. Additionally, the runtime performance was poor similar to the other DP codes which was the main reason that this version of the code was not further pursued and the results will not be further discussed here.

Multi-GPU Code with the basin hopping dataset

In Figs. 4.18 and 4.19, generated structures during the training process of a WGAN with an embedding space of dimension 50 and 100 using the basin hopping data set are illustrated, respectively. Looking at both training processes, the generated structures show significant resemblance with the real structures because the image areas for the vacuum layer and the atomic densities can be clearly distinguished. Furthermore, the atomic densities consist of one or two pixels of high intensity surrounded by a few pixels of smaller intensities. The atomic densities in both channels start to form regular patterns which indicates an order of the structure. Most importantly, the O and Ru channel share the same area for the atomic densities but the atomic densities themselves are placed shifted to each other and do not overlap on the same positions. This is an important feature indicating how realistic the generated structures are because it is impossible for two different atoms to be placed at the same spatial location in a real structure.

Previous calculations with similar hyperparameters but with smaller batchsizes resulted in mode collapses, thus we conclude that here, a larger amount of data that is passed through the WGAN before each parameter update improves the performance and learning ability drastically and has the potential to stabilize the training process.

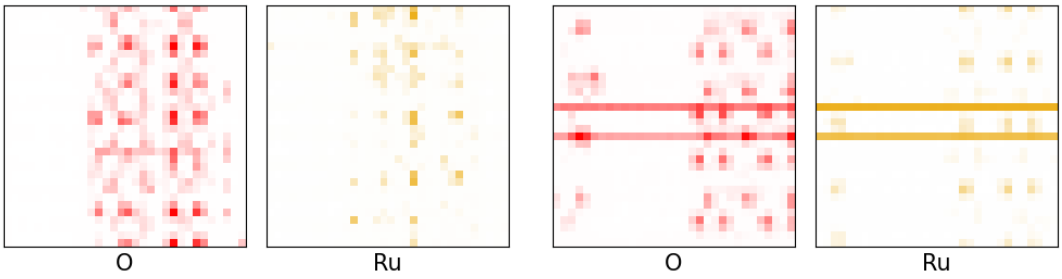


a) Epoch 970 **b)** Epoch 1170

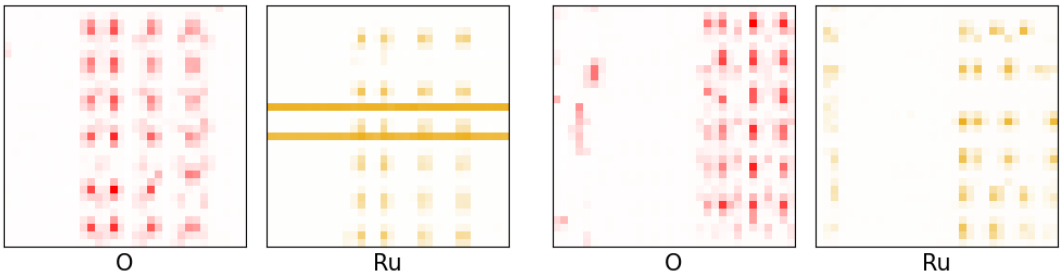
c) Epoch 3580 **d)** Epoch 5220

Figure 4.18: Generated structures from the multi-GPU training process of the energy embedding 2D-DCWGAN with basin hopping dataset, batchsize 2048 and embedding layer size 50.

Only looking at these features, we could conclude that the training process was successful with respect to the geometric information in the O and Ru channels. Nevertheless, in some training structures and in a majority of training structures for the embedding layer size 50 and 100, respectively, we found two horizontal stripes located in the middle of the Ru images. As mentioned above, we assume that the larger batchsize is the significant factor in stabilizing the training set. However, as already discussed, a larger batchsize corresponds to a lower learning rate[74] because fewer parameter updates and thus fewer optimizer steps on the loss surface are performed. Furthermore, the optimizer steps appear less accurate because the single losses of each structure are averaged over a larger batchsize.



a) Epoch 500 **b)** Epoch 1590



c) Epoch 3690 **d)** Epoch 4830

Figure 4.19: Generated structures from the multi-GPU training process of the energy embedding 2D-DCWGAN with basin hopping dataset, batchsize 2048 and embedding layer size 100.

The existence of the stripes in both generator outputs during the training process indicates that the training process is not yet converged. Looking at the generator and critic loss diagrams over the epochs in Figs. 4.20 and 4.21 for embedding layer size 50 and 100, respectively, this assumption is confirmed. Both generator losses are still oscillating and are still increasing towards Epoch 5000. The critic loss in both cases shows small oscillations around zero. This could be problematic because then it leads to the problem of vanishing gradients meaning that the generator is not capable of improving because of the shrinking feedback from the critic. Which could be the reason why the horizontal stripes can be found consistently in both generated sets during the training process.

Another interesting feature in the loss diagrams is the amplitude of the generator loss. For the embedding layer size of 100 it is twice as large as for the embedding layer size of 50 for both the maximum loss at around Epoch 800 and the average loss between Epoch 4000 and 5000. We thus conclude that the doubled embedding dimension is costly in terms of the optimization process and could also explain why the mitigated mode collapse consisting of horizontal stripes for the doubled embedding dimension is stronger.

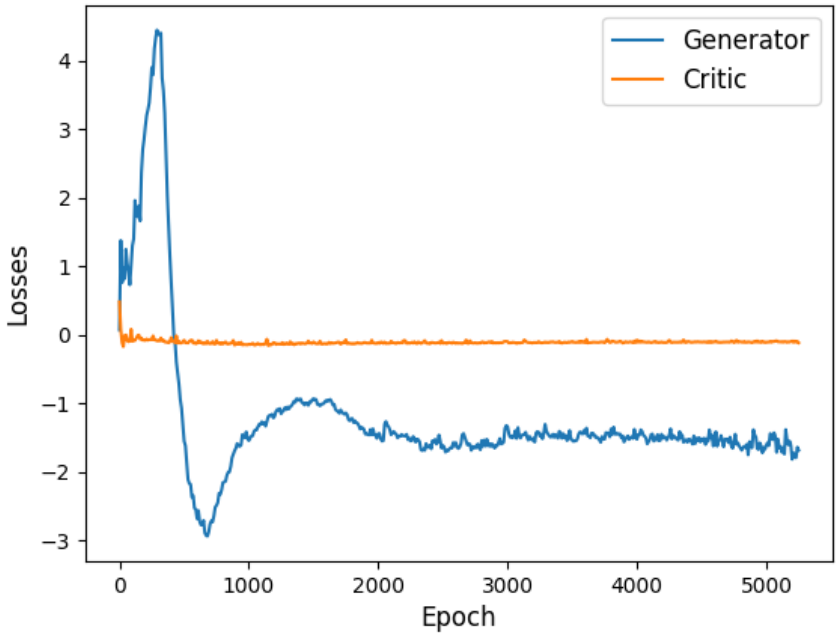


Figure 4.20: Losses during the training process of the energy embedding 2D-DCWGAN with basin hopping dataset, batchsize 2048 and embedding layer size 50.

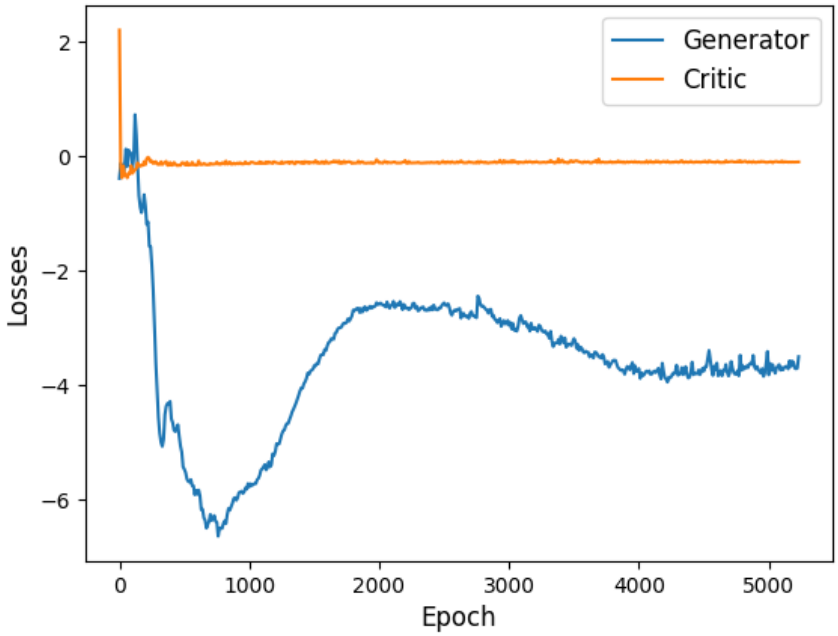
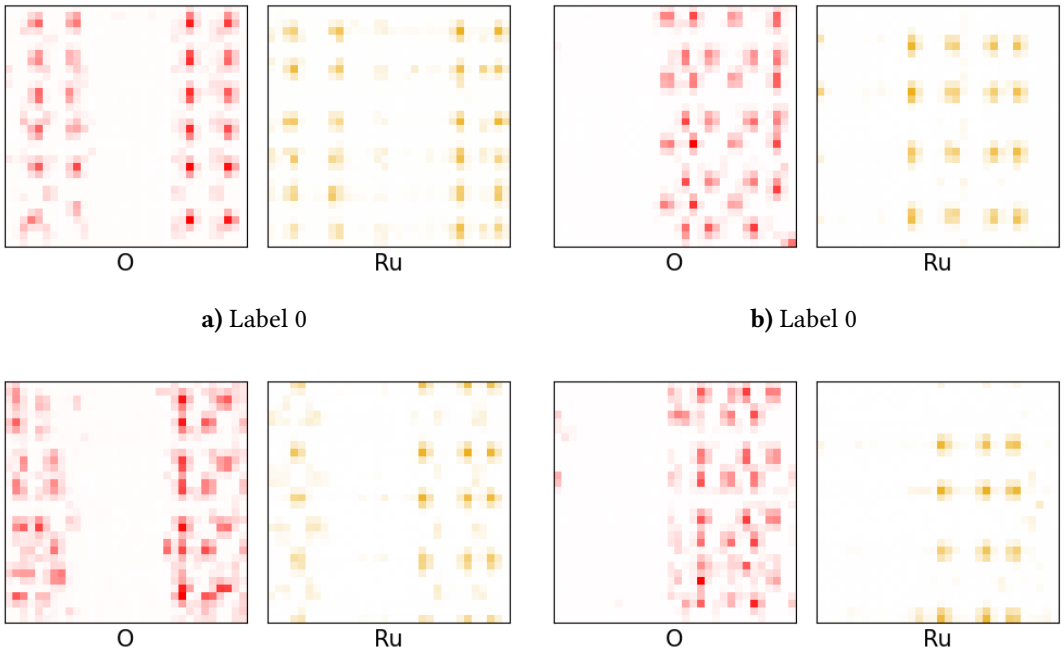


Figure 4.21: Losses during the training process of the energy embedding 2D-DCWGAN with basin hopping dataset, batchsize 2048 and embedding layer size 100.

To further investigate the effect of the embedding layer size, we want to compare the generator output for fixed labels. In Fig. 4.22, generated structures are illustrated for labels 0 and 9 for a WGAN with energy embedding layer size 50. Here, similar structures for label 0 and label 9 for two representative examples of the random generator structures are compared. All of them show decent geometric quality comparable to the ones in Fig. 4.22, the assignment to distinct geometric features or patterns, however, is not represented in the dataset meaning. Hence, all types of structures can be found for all labels at some point.



a) Label 0 **b)** Label 0

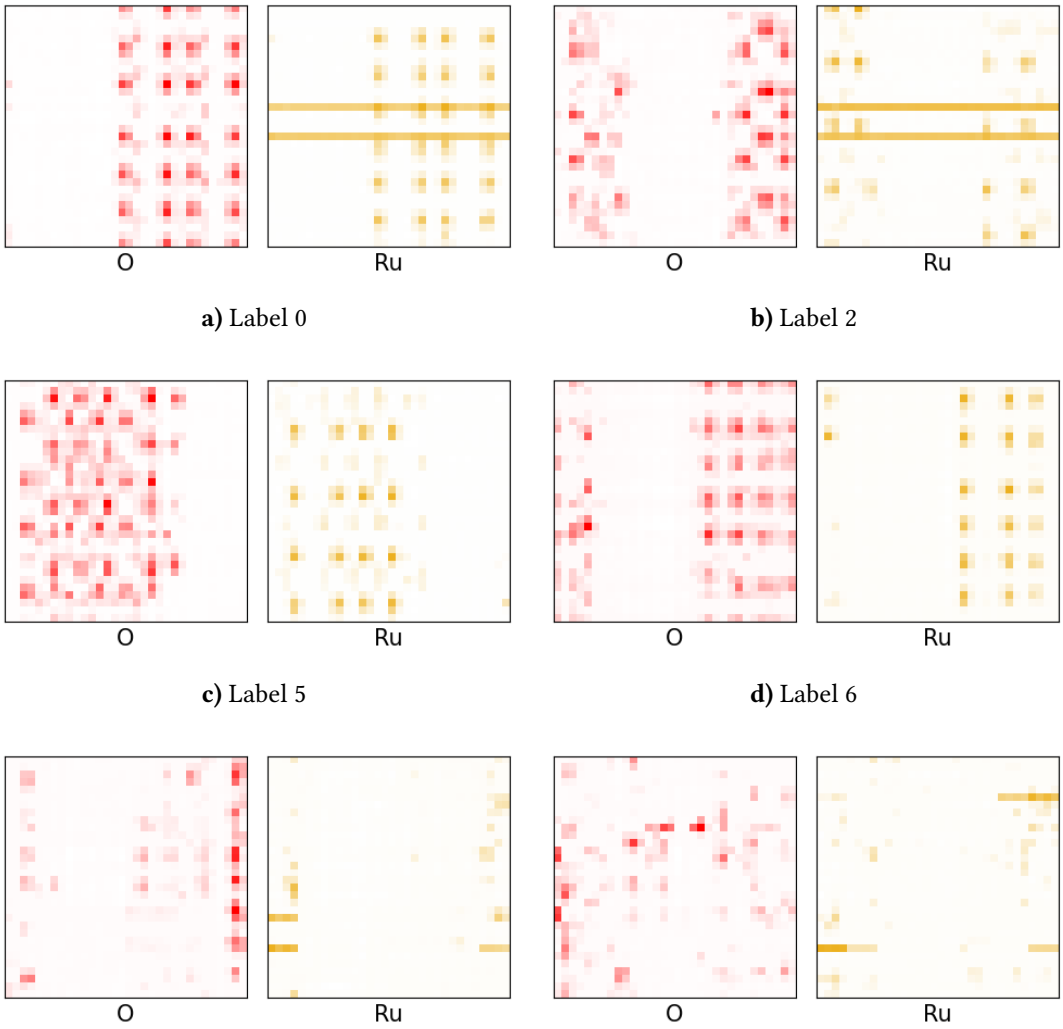
c) Label 9 **d)** Label 9

Figure 4.22: Generated structures produced with fixed labels by the until Epoch 5240 trained generator of the energy embedding 2D-DCWGAN with basin hopping dataset, batchsize 2048 and embedding layer size 50.

This is different for the generated structures for a WGAN with energy embedding layer size 100, illustrated in Fig. 4.23. Without any exceptions, all generated structures with labels 0, 1 and 2 show the horizontal stripes which we interpret as a mitigated mode collapse that the generator encodes with this part of the latent space. Labels 5 and 6 produced high quality images, similar to Fig. 4.22. Whereas for labels 8 and 9, the generator outputs uncorrelated pixels that show no clear or distinct features. The fact that the generator connects these patterns with labels 8 and 9 is a further proof that the training process is not converged for the larger embedding dimension at the same number of epochs.

Thus, the question arises if the larger embedding dimension opens the door for the generator to assign different geometric features better to different latent space encoded vectors or if this is just a relict coming from training divergency? Looking at the two-dimensional

problem, we already analyzed that the information between structures and energy class labels is distorted upon slicing the three-dimensional structures and feeding them separately to the WGAN. It is therefore more likely that the trained generator with the smaller embedding dimension is further converged in the training process than its counterpart. So, its learned connection between class labels and structures is random because in 2D, there is no proper structure-energy-relationship.



a) Label 0 **b)** Label 2

c) Label 5 **d)** Label 6

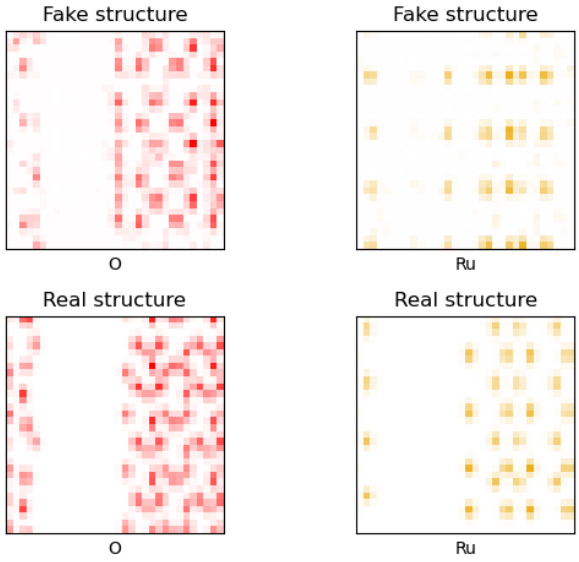
e) Label 8 **f)** Label 9

Figure 4.23: Generated structures produced with fixed labels by the until Epoch 5240 trained generator of the energy embedding 2D-DCWGAN with basin hopping dataset, batchsize 2048 and embedding layer size 100.

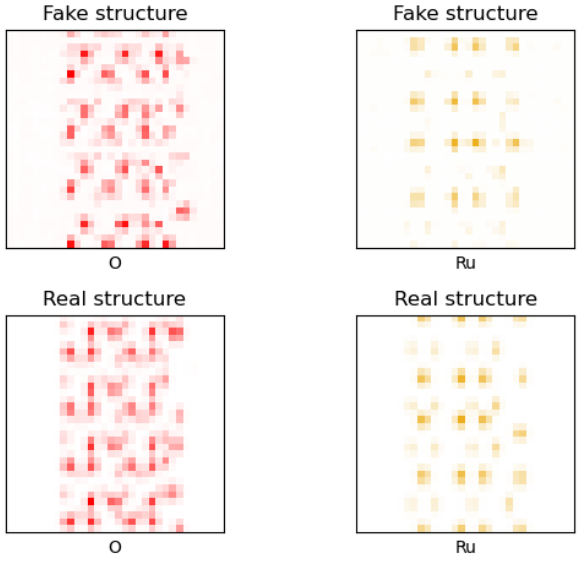
Both embedding layer sizes output decent geometric images, it is only a question of computational efficiency to have a faster converging WGAN. On the other hand, when proceeding to three-dimensional inputs, the structures exhibit more data points. It is thus to explore

whether or not a bigger embedding dimension is required to have the capacity to properly learn structure-energy-relationships.

Since the generated structures during the training process were of high-quality, the question of overfitting arises. As for the *Vanilla* 2D-DCWGAN, we computed the SSIM for all generated structures and compared them with the database. For the WGAN with embedding layer size 50, we found 24 of 524 generated structures were similar to the database and for an embedding layer size 100, we only found 21 of 524 generated structures similar. In Fig. 4.24 and 4.25, two representative examples for similar fake and real structures are illustrated. As before, they exhibit similar cell arrangement, but the details are different. We can thus confidently conclude that our energy embedding 2D-DCWGAN framework is on the hand producing samples that are close to the real dataset and on the other hand not memorizing the dataset because the generated samples can still be distinguished from the dataset.

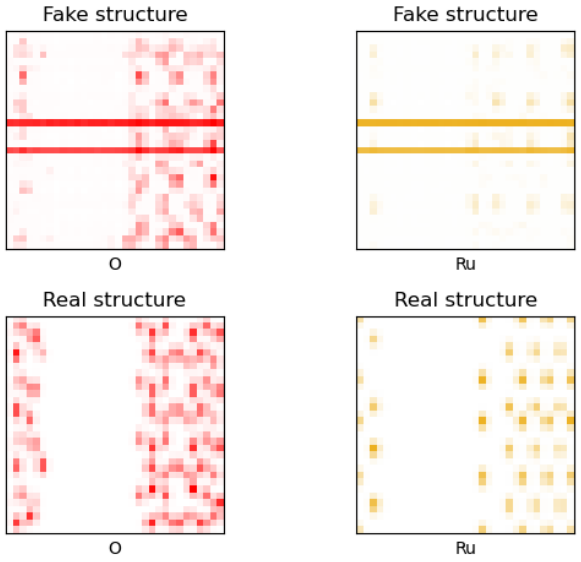


a)

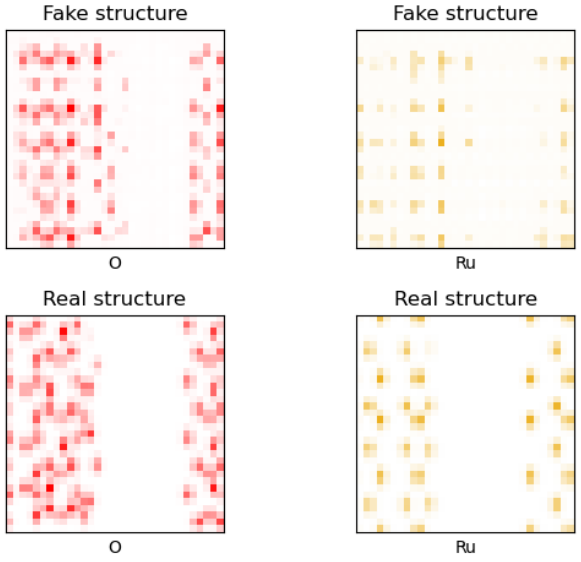


b)

Figure 4.24: Comparison of generated training structures (fake structures) with dataset structures (real structures) to check for overfitting in the multi-GPU Training process of the energy embedding 2D-DCWGAN with basin hopping dataset, batchsize 2048 and embedding layer size 50. The real structures were assigned to the fake structures based on their SSIM[79] value.



a)



b)

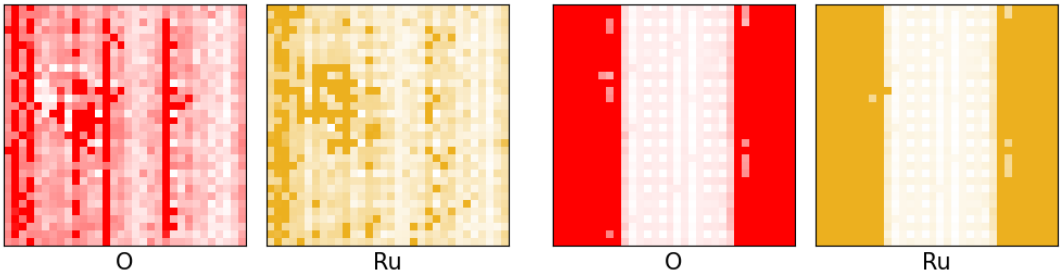
Figure 4.25: Comparison of generated training structures (fake structures) with dataset structures (real structures) to check for overfitting in the multi-GPU Training process of the energy embedding 2D-DCWGAN with basin hopping dataset, batchsize 2048 and embedding layer size 100. The real structures were assigned to the fake structures based on their SSIM[79] value.

4.1.5 Lattice Regressor 2D-DCWGAN

In Section 3.3.3, we introduced two auxiliary networks to our 2D-DCWGAN to predict lattice lengths. In the following, we will discuss the results of the lattice regressor 2D-DCWGAN and the evolution of this computational architecture from a single-GPU version, to the DP multi-GPU and finally to the DDP multi-GPU version.

Single-GPU code with non-extended GAP dataset

Our first attempt in the single-GPU code for batchsize 32 and the GAP dataset with non-extended cells (see Fig. 3.7) resulted in mode collapses. The regressed lattice lengths were exploding to infinite values or to zero. In contrast to the previously shown mode collapses, this mode collapse, as illustrated in Fig. 4.26, consists of three vertical stripes, two of them with one consistent high intensity and the middle stripe with some rectangular arranged pixels of weak intensity. As pixels of high intensity correspond to a higher activation of neurons within the network, we assume that the WGAN here learns predominantly the features of the cell background rather than the atomic densities. These findings were our main motivation to extend the cells in the GAP dataset.



a) Epoch 1360 **b)** Epoch 2260

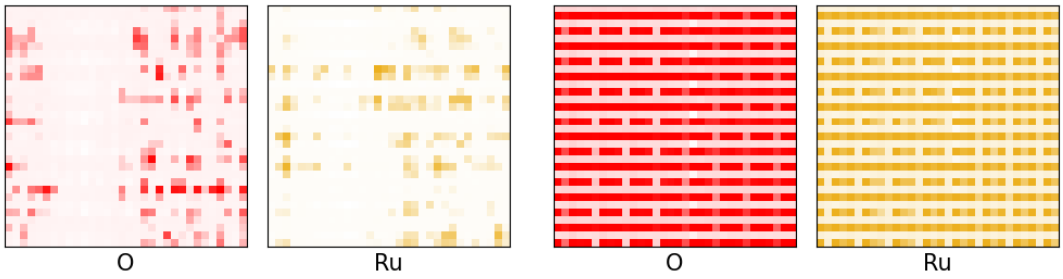
Figure 4.26: Generated structures from the single-GPU training process of the lattice regressor 2D-DCWGAN with non-extended GAP dataset and batchsize 32 at different epochs.

Multi-GPU with extended GAP dataset

The following two multi-GPU simulations based on the DP package were trained with the extended cell GAP datasets (see Fig.3.7). As illustrated in Figs. 4.27 and 4.28, the generator output for both extended GAP datasets improved significantly in contrast to the previous simulation (see Fig.4.26) for early epochs. In the beginning of the training process, the background is presented with pixels of low intensity and atomic positions start to be resembled through some grouped pixels. The overall image quality, however, is still low because the Gaussian density shape is not yet reached and the positions for the atomic positions is still uncoordinated and no slab-like structure can be recognized. Interestingly, after about 300 epochs, both simulations result in a mode collapse. This time, the mode collapse resembles the mode collapses from other simulations as horizontal stripes. We thus conclude that the dataset design is a crucial ingredient for a successful training process, especially for more complicated computational architectures. For these simulations, the other remaining hyperparameters like the batchsize, learning rates or the overall dataset size still have to be optimized.

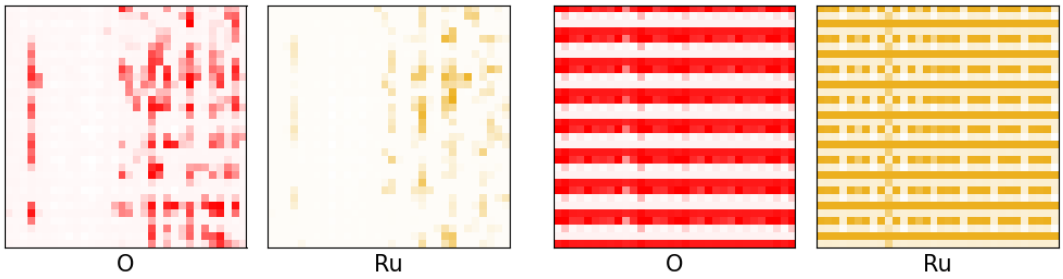
No physically meaningful lattice lengths could be deduced from these computations yet. In the beginning of the training process, the ratio of the lattice lengths was 1:1 meaning that the generator was yet not capable of extracting information from the generated images. After the mode collapse, the predicted lattice lengths tend towards zero for a majority of the

generated structures. Since the mode collapse prevents the WGAN from extracting sensible lattice lengths, we cannot deduce the potential of this computational architecture for this code version.



a) Epoch 110 **b)** Epoch 260

Figure 4.27: Generated structures from the multi-GPU training process of the lattice regressor 2D-DCWGAN with extended 148 structure GAP dataset and batchsize 32 at different epochs.



a) Epoch 60 **b)** Epoch 510

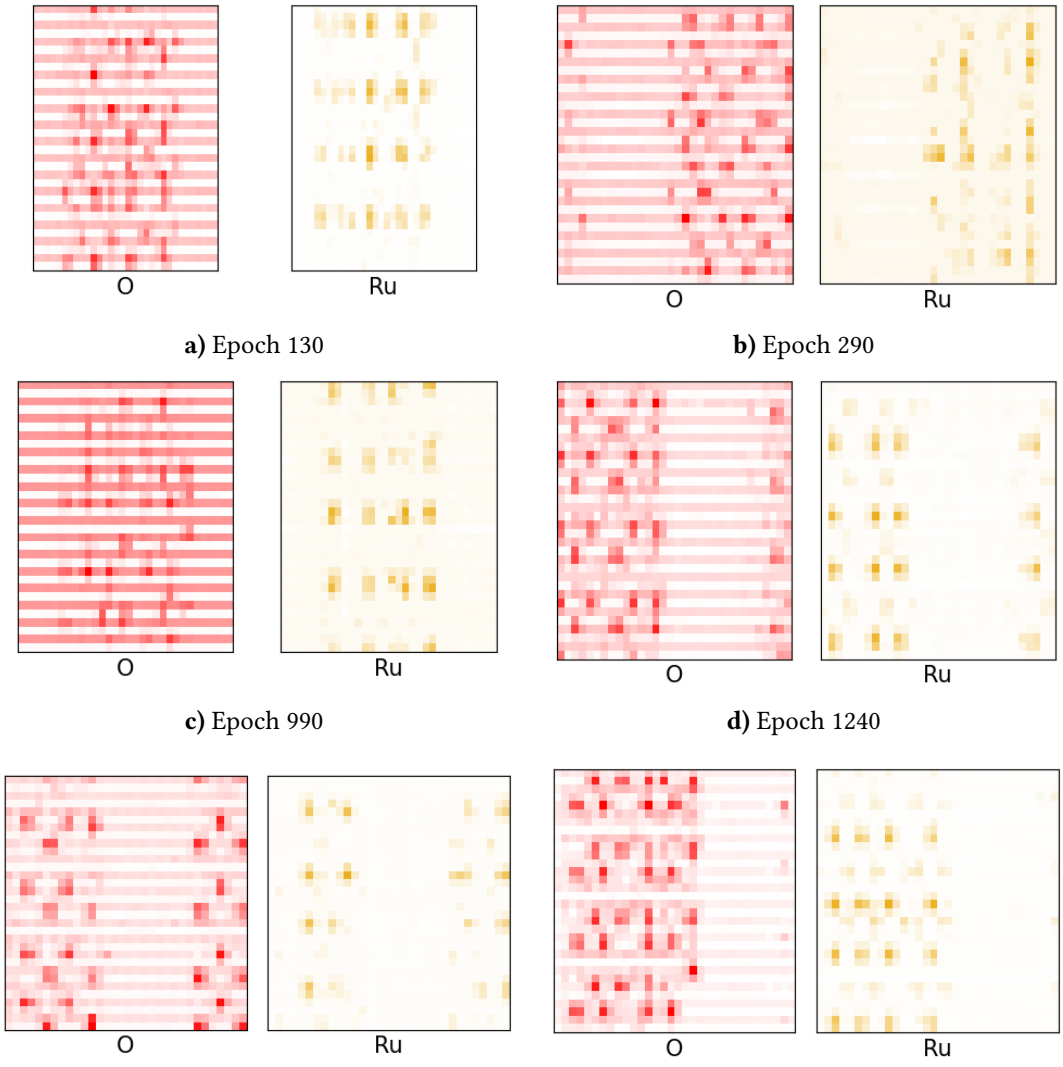
Figure 4.28: Generated structures from the multi-GPU training process of the lattice regressor 2D-DCWGAN with extended 91 structure GAP dataset and batchsize 32 at different epochs.

Multi-GPU Code with basin hopping dataset

With aid of the DDP code version, simulations incorporating the basin hopping dataset and batchsizes 512, 1024 and 2048 were started. All three simulations lead to similar geometric outcomes. The regressed lattice length for batchsize 1024 delivered the best results and will be thus discussed in the following.

In Fig. 4.29, generated structures from the training process of the lattice regressor 2D-DCWGAN with batchsize 1024 are illustrated. At first sight, all images exhibit horizontal stripes in the background that we interpret as a mitigated form of a mode collapse. This seems to be a typical feature emerging in the training process if the simulations are not converged yet. Apart from the horizontal stripes, all images consist of areas with pixels of high intensity that resemble the Gaussian shaped atomic densities. These densities are ordered and resemble the structures from the real dataset, especially for later epochs from 1500 onward. The quality

of the geometric output improves over time not only by more defined Gaussian densities but also by a weakening of the horizontal stripes in the background. This trend can be captured in the corresponding loss diagram in Fig. 4.30. Between Epoch 1200 to 4000, the absolute value of the generator loss continuously decreases. This can be a sign that the generator slowly learns to adjust its structures by adjusting the weights such that the horizontal stripe feature has less influence.



a) Epoch 130 **b)** Epoch 290

c) Epoch 990 **d)** Epoch 1240

e) Epoch 2060 **f)** Epoch 3250

Figure 4.29: Generated structures from the multi-GPU training process of the lattice regressor 2D-DCWGAN with basin hopping dataset and batchsize 1024 at different epochs. The lattice images were rescaled using the predicted lattice length ratios.

Another special feature of Fig. 4.29 is that we used the predicted lattice length ratios to scale the images upon plotting. In these images, it is visible that the generator starts to learn a realistic ratio between the x/y-directions and the z direction which is the aim of this computational approach. In the beginning of the training process, like in Epoch 130 or 290, the lattice

length were even more characteristic. Over the course of the training process, the generator starts to unlearn this feature again, as seen in Epochs 2060 and 3250 in Fig. 4.29 the ratio of the lattice lengths start to strive towards 1:1. This is true for most generated samples in this time range. In this case, we assume that the lattice loss is not weighted strongly enough and its contributions thus vanishes over the course of the epochs. In Eq. (3.5), we introduced the hyperparameter γ and set it to 0.1 for all simulations. For future simulations, we want to increase γ to verify if a larger influence of the lattice loss on the overall loss function of the generator can improve the learning process of the lattice lengths.

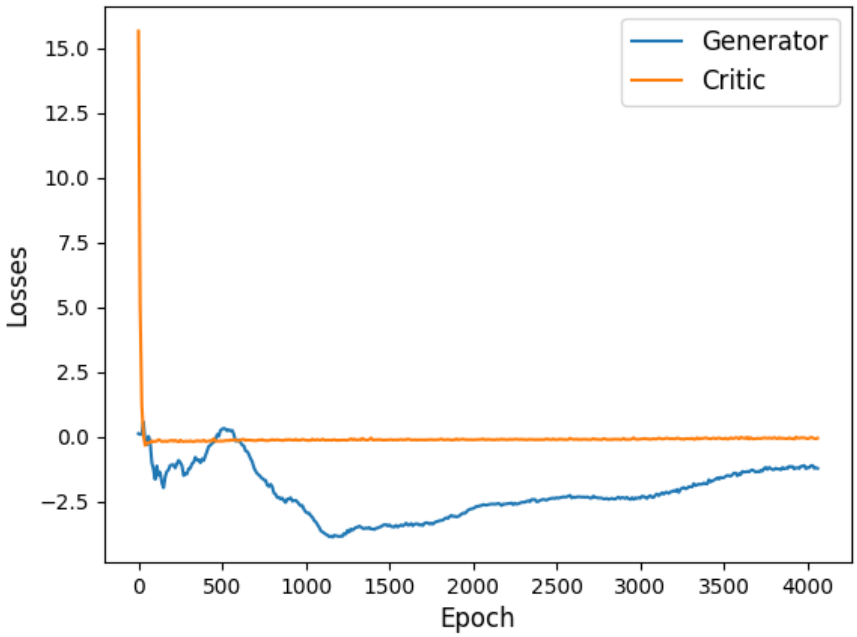


Figure 4.30: Losses during the training process of the lattice regressor 2D-DCWGAN with basin hopping dataset and batchsize 1024.

4.2 3D-DCWGAN

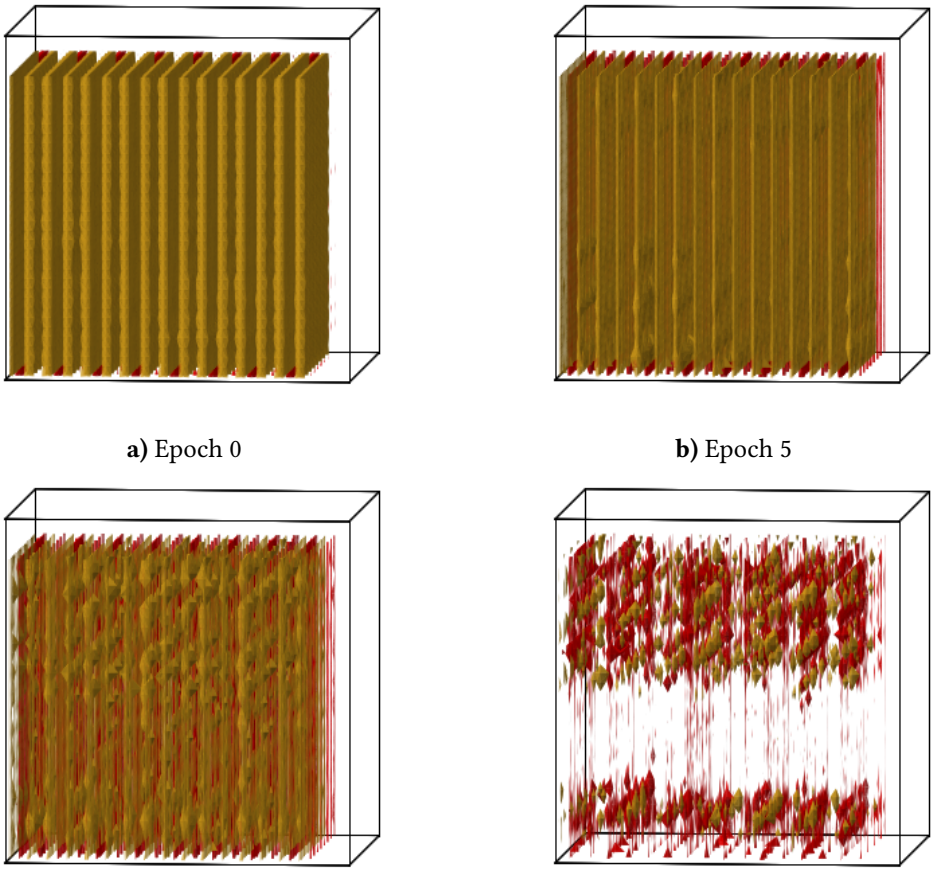
Due to the increase of computational runtime for a three-dimensional framework, the main aim of the simulations on two-dimensional computational architectures was to determine stable hyperparameters that can now be applied to the three-dimensional computational architectures with realistic three-dimensional structures. We will now introduce the *Vanilla* 3D-DCWGAN.

4.2.1 *Vanilla* 3D-DCWGAN

The computational architecture of the *Vanilla* 3D-DCWGAN and the current hyperparmater settings were introduced in Section 3.3.4. In the following, we will focus on the learning process of our *Vanilla* 3D-DCWGAN.

In Fig. 4.31, four generated density-based structures from the training process of the *Vanilla* 3D-DCWGAN are illustrated from the beginning of the training process. In Epoch 0, the structure resembles a three-dimensional version of the mode collapse consisting of vertical planes for both, the Ru and O channels. In Epoch 5 and 15, the first development of the learning process is visible in which the solid planes start to dissolve and to rather form assembled groups of voxels. In Epoch 35, the first time a density-based structure emerges with a clear separation between atomic densities and a vacuum layer area. This learning process is visible in the loss diagram in Fig. 4.32. The generator loss decreases to -54 and reaches its peak value already in the first few epochs. After Epoch 35, the absolute value of the generator loss starts to decrease. Over the course of epochs, the critic shows oscillations with a smaller amplitude than the generator. After Epoch 50, the generator starts to show oscillation of smaller amplitude. This indication of the progressing learning success is also found in later sampled generated structures. The sample quality for Epoch 220 and Epoch 280 in Fig. 4.33 has drastically improved compared to the structures from the training start (see Fig. 4.31).

To further evaluate the sample quality, we sliced the density-based structures from Epoch 35, 220 and 280 in Fig. 4.34. Here, the slices for each epoch were taken in two different directions at the same position. For Epoch 35, the horizontal stripes are the dominant feature and only some single pixels of high-intensity are seen that do not resemble Gaussian densities yet. For Epoch 220, the stripe feature is still present, but the atomic densities in the background of the cell start to build. For Epoch 280, again, the horizontal stripes are a dominant feature, however, the atomic densities for three of four channels stand out and start to assemble in a ordered fashion. Furthermore, distinct areas for the vacuum layer and the cell area emerge. In combination with the ongoing oscillation in the loss diagram, for both the generator and the critic, we can confidently conclude that the *Vanilla* 3D-DCWGAN is still in the learning process and improving. As the image quality is not perfect yet and the horizontal stripes are still present, we will continue to converge this simulation. It can already be interpreted as a proof of concept that the computational architecture for our *Vanilla* 3D-DCWGAN is capable of learning the basic features of our three-dimensional density-based RuO_2 input.



a) Epoch 0 **b)** Epoch 5

c) Epoch 15 **d)** Epoch 35

Figure 4.31: Three-dimensional density-based structure output during the training process of the *Vanilla* 3D-DCWGAN with the basin hopping data set at different epochs. Ru atoms are drawn as yellow spheres and O atoms as red spheres.

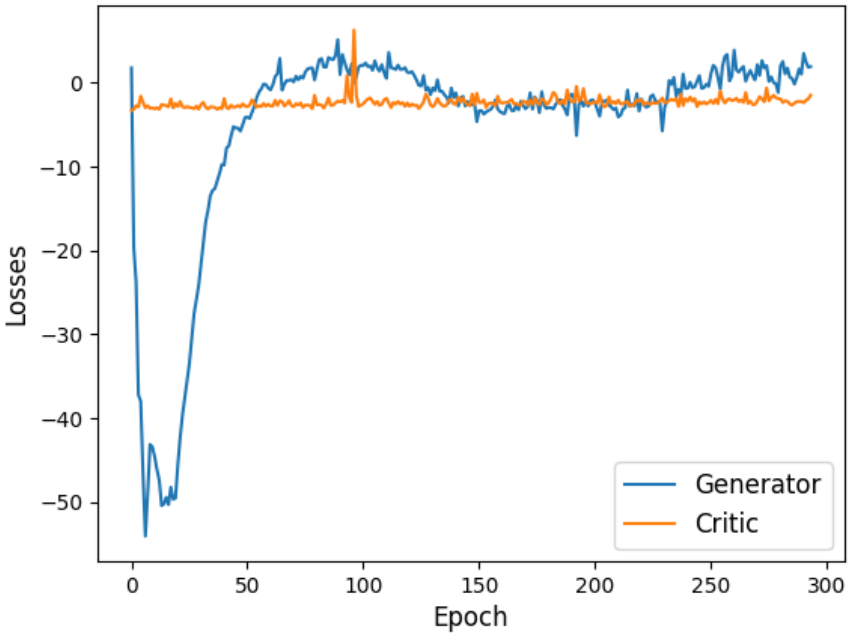
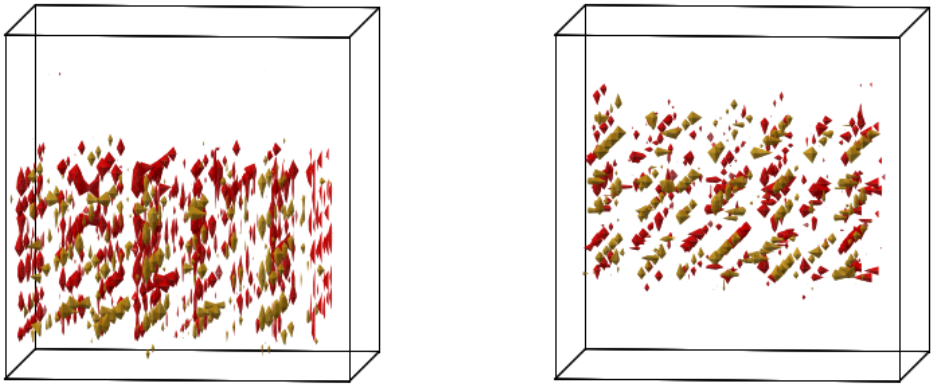
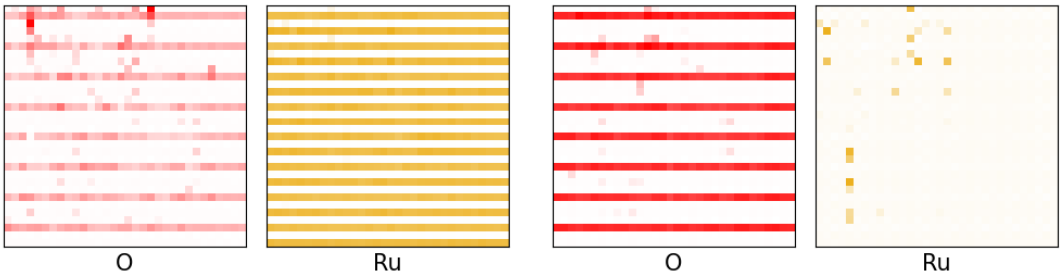


Figure 4.32: Losses during the training process of the *Vanilla* 3D-DCWGAN with basin hopping dataset and batchsize 32.

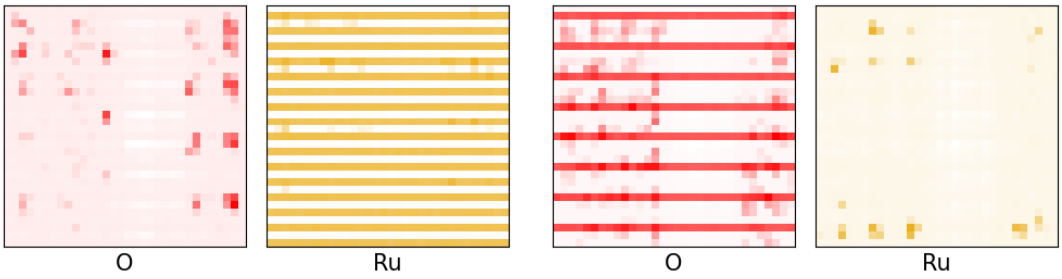


a) Epoch 220 **b)** Epoch 280

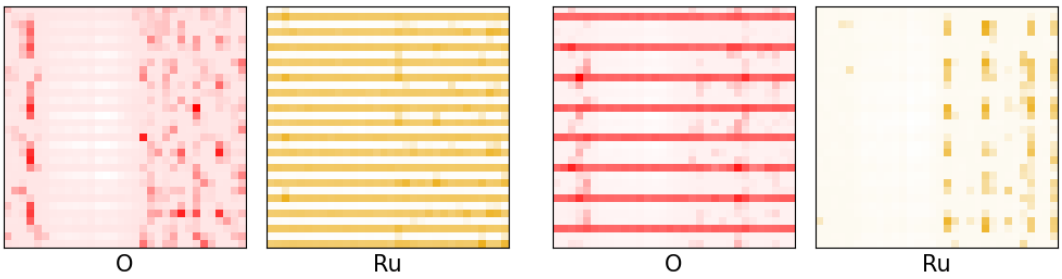
Figure 4.33: Three-dimensional density-based structure output during the training process of the *Vanilla* 3D-DCWGAN with the basin hopping data set at different epochs. Ru atoms are drawn as yellow spheres and O atoms as red spheres.



a) Epoch 35 **b)** Epoch 35



c) Epoch 220 **d)** Epoch 220



e) Epoch 280 **f)** Epoch 280

Figure 4.34: 2D slices of three-dimensional structures from the multi-GPU training process of the *Vanilla* 3D-DCWGAN with basin hopping dataset and batchsize 32 at different epochs. The slices are taken from the xy and yz direction for each epoch.

5 Conclusion and Outlook

The main aim of this thesis was to train a generative model based to explore the PES of RuO_2 in a thorough manner and to cheaply generate new structural estimates for possible surface terminations. To stabilize the network training process, two different training sets, convolutional network layers and a softer measure for the loss function were implemented and discussed.

In the first part of this study, the training efficiency was enhanced by a dimension reduction of the training set size. This enabled to efficiently perform a hyperparamter search and investigate three different approaches, one to solely create new structure outputs and two other approaches to combine structural information together with energy labels or lattice lengths. For the *Vanilla* 2D-DCWGAN, our main finding was that the image quality of the atomic densities and thus the learning performance strongly depends on the dataset design and the total amount of input structures in the training set. Since the structure-energy relationship is distorted for two-dimensional slices from three-dimensional structures, the latent space encoding for our energy class labels - integer-based or one hot encoded - can only serve as a proof of principle that has to be addressed in further studies in a three-dimensional framework. Augmenting two auxiliary networks to embed physical quantities such as the lattice lengths in our DCWGAN approach has proven potential that has to be further explored with a higher lattice loss penalty during the training process in a three-dimensional framework as well. When embedding extra information in channels or through latent space encoding, it is important to find the sweet spot such that the additional information is not overwriting the density-based data but also that its implementation has a significant influence on the outcome of the training process. To avoid mode collapse, tailored input data, tuned learning rates, adjusted batchsizes and sizes for the latent space encoding vectors were crucial for the training success in all simulations.

Upgrading our code with the DDP[67] package significantly enhanced the training process and we highly recommend single-node multi-GPU based training with this package in PyTorch for other scientists handling larger datasets. The unlocked computational speed enabled us to process all 28,903 structures in the three-dimensional DCWGAN framework such that 80 epochs can run within 24 hours. In this learning process, the *Vanilla* 3D-DCWGAN started to converge. Further simulation are needed to converge the three-dimensional generator output and to tune the hyperparameters required for the mixed three-dimensional architectures. Ultimately, we want to develop a framework that simultaneously generates structures based on a defined energy range and with a realistic lattice length prediction.

With this study, we aimed to create a powerful tool to unravel the longstanding puzzle of catalytically relevant RuO_2 surface structures. The developed two-dimensional DCWGAN framework forms the first steps towards the output of realistic three-dimensional structure

outputs that can be used as inputs for a subsequent geometry optimization. With this deep generative approach, we aim to bridge the materials gap for theoretical studies by enabling a large scale search on the chemical space of RuO_2 surface structures and by finding novel structures that can close the knowledge gap for some surface processes. As our framework was restricted to two-dimensional density-based images, we found novel two-dimensional structures that showed a strong resemblance with the original dataset while featuring different details for the density distributions. We are thus confident that our three-dimensional DCWGAN framework has similar potential to generate new structures.

The next implementation steps for further studies are thus to optimize the three-dimensional DCWGAN such that converged structures without mode collapse can be generated. Another goal for realistic three-dimensional structures is to tune the hyperparameters for the three-dimensional lattice regressor networks such that realistic structure lengths are available to permit an accurate backmapping. Once realistic three-dimensional structures are achieved, we aim to steer the latent space design such that the generated structures corresponds to a certain energy range by either using integer-based or one hot encoded conditioning vectors or by embedding the class labels in an additional embedding layer in the network architecture. A further idea would be to use the class labels not only for the energy range of the structures but for example for the oxygen chemical potential under which they were sampled. This allows not only the introduction of energetics into our deep generative model but also to incorporate thermodynamical information.

Considering the training process of the neural network, a better understanding of the implications of different hyperparameters can be achieved by sampling different latent space dimensions Z. It has been shown in [54] that the continuity of the GAN mapping and thus its convergence is significantly influenced by the dimension of the easier probability distribution P_Z. In addition, gradual changes of other hyperparameters during the training process can be introduced to further speed up and stabilize training. Slowly increasing the batchsize during the learning process was found to have a similar effect on the convergence process as simulated annealing.[74] It would be interesting in three dimensions to start with a lower batchsize and then increase it until the maximum value of 32 and then to compare the convergence behavior of the generated structures. Furthermore, to enlarge the structural variety of the explored chemical space, the dataset design is crucial. As the basin hopping set only included (001) surface terminations, we can obtain a more divers database by adding further terminations. Extensive basin hopping sampling for the dataset generation can become a small computational bottleneck, however as the algorithm is fully automatized, these computations can be performed on-the-fly.

When training the WGAN, it is not only important to tune the hyperparameters and the dataset design, but also to understand the learning process. As neural networks are quite complex on the level of individual neurons, the application of tools that help to understand the underlying processes such as distribution statistics or perform gradient analysis during the learning process can provide relevant insights.[71] The GradCAM[71] package from Captum offers interesting approaches to link activations in the network to corresponding areas in the data

input. This can be crucial to understand the classification criteria for decision-making process of the critic. It can, on one hand, help to tune the dataset design, by filtering relevant areas in the dataset and highlighting features used to distinguish real and fake data. On the other hand, the obtained evolving of features provides insights on depending on the hyperparameters of the generator. When it comes to explainable artificial intelligence, another point is crucial: understanding what the generator actually learns from the real data distribution is highly important. If the generator only learns the features from the bulk or the relaxation layers in our basin hopping dataset and fails to learn the specific features of the inserted oxygen atoms in the larger supercell, an important part of the chemical space would not be covered. For this purpose, semantic segmentation networks[80] can be used to investigate the relationship between the input data and the features of the output data. In this approach, the generator layers are inverted to investigate which data features are learned. This can i.e. provide insights on dropped modes from the input data.[80]

After the performance of the training process is improved and our implementation goals are achieved, we can proceed to the task of backmapping the structures and to investigate the novelly generated surface terminations. Novel structures can be identified and compared to existing ones with the kernel similarity measurement method from [18] in which the SOAP[61] kernels are used for structural comparison. Once novel structures are obtained, their stability under reaction conditions can be determined in ab-initio thermodynamic approaches. These novel structures can then help to understand the catalytically active structures of RuO_2 in the CO oxidation reaction and can potentially help to elucidate on the reaction mechanism.

Bibliography

[1] A. Ananth, R. H. Jeong, and J.-H. Boo, "Preparation, characterization and co oxidation performance of ag2o/γ-al2o3 and (ag2o+ ruo2)/γ-al2o3 catalysts", Surfaces **3**, 251–264 (2020).

[2] D. G. Vlachos and S. Caratzoulas, "The roles of catalysis and reaction engineering in overcoming the energy and the environment crisis", Chemical Engineering Science **65**, 18–29 (2010).

[3] N. Soliman, "Factors affecting co oxidation reaction over nanosized materials: a review", Journal of Materials Research and Technology **8**, 2395–2407 (2019).

[4] T. Barakat, J. C. Rooke, E. Genty, R. Cousin, S. Siffert, and B.-L. Su, "Gold catalysts in environmental remediation and water-gas shift technologies", Energy & Environmental Science **6**, 371–391 (2013).

[5] V. Narkhede, J. Aßmann, and M. Muhler, "Structure-activity correlations for the oxidation of co over polycrystalline ruo2 powder derived from steady-state and transient kinetic experiments", Zeitschrift für Physikalische Chemie **219**, 979–995 (2005).

[6] S. Matera, M. Maestri, A. Cuoci, and K. Reuter, "Predictive-quality surface reaction chemistry in real reactor models: integrating first-principles kinetic monte carlo simulations into computational fluid dynamics", Acs Catalysis **4**, 4081–4092 (2014).

[7] J. E. Sutton, J. M. Lorenzi, J. T. Krogel, Q. Xiong, S. Pannala, S. Matera, and A. Savara, "Electrons to reactors multiscale modeling: catalytic co oxidation over ruo2", ACS Catalysis **8**, 5002–5016 (2018).

[8] D. Rosenthal, F. Girgsdies, O. Timpe, G. Weinberg, and R. Schlögl, "Oscillatory behavior in the co-oxidation over bulk ruthenium dioxide—the effect of the co/o2 ratio", Zeitschrift für Physikalische Chemie **225**, 57–68 (2011).

[9] D. Rosenthal, F. Girgsdies, O. Timpe, R. Blume, G. Weinberg, D. Teschner, and R. Schlögl, "On the co-oxidation over oxygenated ruthenium", Zeitschrift für Physikalische Chemie **223**, 183–208 (2009).

[10] A. Böttcher, H. Niehus, S. Schwegmann, H. Over, and G. Ertl, "Co oxidation reaction over oxygen-rich ru (0001) surfaces", The Journal of Physical Chemistry B **101**, 11185–11191 (1997).

[11] H. Over, M. Knapp, E. Lundgren, A. Seitsonen, M. Schmid, and P. Varga, "Visualization of atomic processes on ruthenium dioxide using scanning tunneling microscopy", ChemPhysChem **5**, 167–174 (2004).

[12] C. H. Peden and D. W. Goodman, "Kinetics of carbon monoxide oxidation over ruthenium (0001)", The Journal of Physical Chemistry **90**, 1360–1365 (1986).

[13] J. Aßmann, D. Crihan, M. Knapp, E. Lundgren, E. Löffler, M. Muhler, V. Narkhede, H. Over, M. Schmid, A. P. Seitsonen, et al., "Understanding the structural deactivation of ruthenium catalysts on an atomic scale under both oxidizing and reducing conditions", Angewandte Chemie International Edition **44**, 917–920 (2005).

[14] A. Böttcher, M. Rogozia, H. Niehus, H. Over, and G. Ertl, "Transient experiments on co2 formation by the co oxidation reaction over oxygen-rich ru (0001) surfaces", The Journal of Physical Chemistry B **103**, 6267–6271 (1999).

[15] T. Wang, J. Jelic, D. Rosenthal, and K. Reuter, "Exploring pretreatment–morphology relationships: ab initio wulff construction for ruo2 nanoparticles under oxidising conditions", ChemCatChem **5**, 3398–3403 (2013).

[16] K. Reuter and M. Scheffler, "Composition and structure of the ruo 2 (110) surface in an o 2 and co environment: implications for the catalytic formation of co 2", Physical Review B **68**, 045407 (2003).

[17] K. Reuter and M. Scheffler, "Composition, structure, and stability of ruo 2 (110) as a function of oxygen pressure", Physical Review B **65**, 035406 (2001).

[18] J. Timmermann, Y. Lee, C. G. Staacke, J. T. Margraf, C. Scheurer, and K. Reuter, "Data-efficient iterative training of gaussian approximation potentials: application to surface structure determination of rutile iro2 and ruo2", The Journal of Chemical Physics **155**, 244107 (2021).

[19] J. Noh, J. Kim, H. S. Stein, B. Sanchez-Lengeling, J. M. Gregoire, A. Aspuru-Guzik, and Y. Jung, "Inverse design of solid-state materials via a continuous representation", Matter **1**, 1370–1384 (2019).

[20] B. Kim, S. Lee, and J. Kim, "Inverse design of porous materials using artificial neural networks", Science advances **6**, eaax9324 (2020).

[21] R. Pollice, G. dos Passos Gomes, M. Aldeghi, R. J. Hickman, M. Krenn, C. Lavigne, M. Lindner-D'Addario, A. Nigam, C. T. Ser, Z. Yao, and A. Aspuru-Guzik, "Data-driven strategies for accelerated materials design", Accounts of Chemical Research **54**, PMID: 33528245, 849–860 (2021).

[22] C. Panosetti, K. Krautgasser, D. Palagin, K. Reuter, and R. J. Maurer, "Global materials structure search with chemically motivated coordinates", Nano letters **15**, 8044–8048 (2015).

[23] K. Krautgasser, C. Panosetti, D. Palagin, K. Reuter, and R. J. Maurer, "Global structure search for molecules on surfaces: efficient sampling with curvilinear coordinates", The Journal of chemical physics **145**, 084117 (2016).

[24] J. J. Pietron, M. B. Pomfret, C. N. Chervin, J. W. Long, and D. R. Rolison, "Direct methanol oxidation at low overpotentials using pt nanoparticles electrodeposited at ultrathin conductive ruo2 nanoskins", Journal of Materials Chemistry **22**, 5197–5204 (2012).

[25] H. Yu, K. Zeng, X. Fu, Y. Zhang, F. Peng, H. Wang, and J. Yang, "Ruo2· x h2o supported on carbon nanotubes as a highly active catalyst for methanol oxidation", The Journal of Physical Chemistry C **112**, 11875–11880 (2008).

[26] E. Riedel and C. Janiak, "Anorganische chemie", in *Anorganische chemie* (de Gruyter, 2022).

[27] D. P. Woodruff and T. A. Delchar, *Modern techniques of surface science*, 2nd ed., Cambridge Solid State Science Series (Cambridge University Press, 1994).

[28] T. E. Madey, H. A. Engelhardt, and D. Menzel, "Adsorption of oxygen and oxidation of co on the ruthenium (001) surface", Surface Science **48**, 304–328 (1975).

[29] A. V. Joshi, *Machine learning and artificial intelligence* (Springer, 2020).

[30] F. E. Harrell, "Regression modeling strategies", Bios **330**, 14 (2017).

[31] J. Hermann, Z. Schätzle, and F. Noé, "Deep-neural-network solution of the electronic schrödinger equation", Nature Chemistry **12**, 891–897 (2020).

[32] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*, `http://www.deeplearningbook.org` (MIT Press, 2016).

[33] J. Schmidhuber, "Deep learning in neural networks: an overview", Neural networks **61**, 85–117 (2015).

[34] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks", in Proceedings of the fourteenth international conference on artificial intelligence and statistics, Vol. 15, edited by G. Gordon, D. Dunson, and M. Dudík, Proceedings of Machine Learning Research (2011), pp. 315–323.

[35] G. Masetti and F. Di Giandomenico, "Analyzing forward robustness of feedforward deep neural networks with leakyrelu activation function through symbolic propagation", in Joint european conference on machine learning and knowledge discovery in databases (Springer, 2020), pp. 460–474.

[36] W. Shang, K. Sohn, D. Almeida, and H. Lee, "Understanding and improving convolutional neural networks via concatenated rectified linear units", in International conference on machine learning (PMLR, 2016), pp. 2217–2225.

[37] S. Ioffe and C. Szegedy, "Batch normalization: accelerating deep network training by reducing internal covariate shift", in International conference on machine learning (PMLR, 2015), pp. 448–456.

[38] J. L. Ba, J. R. Kiros, and G. E. Hinton, *Layer normalization*, 2016.

[39] K. P. Murphy, *Machine learning: a probabilistic perspective* (MIT press, 2012).

[40] G. Schay, *Introduction to probability with statistical applications* (Birkhäuser, 2016).

[41] K. Janocha and W. M. Czarnecki, "On loss functions for deep neural networks in classification", arXiv preprint arXiv:1702.05659 (2017).

[42] *Pytorch documentation multilabelsoftmarginloss*.

[43] G. Lan, *First-order and stochastic optimization methods for machine learning* (Springer Nature, 2020).

[44] D. P. Kingma and J. Ba, *Adam: a method for stochastic optimization*, 2017.

[45] S. Albawi, T. A. Mohammed, and S. Al-Zawi, "Understanding of a convolutional neural network", in 2017 international conference on engineering and technology (icet) (2017), pp. 1–6.

[46] V. Dumoulin and F. Visin, *A guide to convolution arithmetic for deep learning*, 2018.

[47] *Stanford lecture cs231n: convolutional neural networks for visual recognition*, 2021.

[48] *Convolutional network figure*, 2021.

[49] A. Anwar, *What is transposed convolutional layer?*, 2020.

[50] L. Ruthotto and E. Haber, "An introduction to deep generative modeling", GAMM-Mitteilungen, e202100008 (2021).

[51] Y. Hong, U. Hwang, J. Yoo, and S. Yoon, "How generative adversarial networks and their variants work: an overview", ACM Computing Surveys (CSUR) **52**, 1–43 (2019).

[52] J. Gui, Z. Sun, Y. Wen, D. Tao, and J. Ye, "A review on generative adversarial networks: algorithms, theory, and applications", CoRR **abs/2001.06937** (2020).

[53] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets", Advances in neural information processing systems **27** (2014).

[54] M. Arjovsky and L. Bottou, "Towards principled methods for training generative adversarial networks", arXiv preprint arXiv:1701.04862 (2017).

[55] M. Arjovsky, S. Chintala, and L. Bottou, *Wasserstein gan*, 2017.

[56] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville, "Improved training of wasserstein gans", Advances in neural information processing systems **30** (2017).

[57] C. Villani, *Optimal transport: old and new*, Vol. 338 (Springer, 2009).

[58] P. König, "Wgan code framework", `https://thgitlab.rz-berlin.mpg.de/koenig/gancode\_thesisversion.git` (2022).

[59] V. L. Deringer, A. P. Bartók, N. Bernstein, D. M. Wilkins, M. Ceriotti, and G. Csányi, "Gaussian process regression for materials and molecules", Chemical Reviews **121**, PMID: 34398616, 10073–10141 (2021).

[60] A. P. Bartök, M. C. Payne, R. Kondor, and G. Csänyi, "Gaussian approximation potentials: the accuracy of quantum mechanics, without the electrons", Physical review letters **104**, 136403 (2010).

[61] A. P. Bartók, R. Kondor, and G. Csányi, "On representing chemical environments", Phys. Rev. B **87**, 184115 (2013).

[62] A. H. Larsen, J. J. Mortensen, J. Blomqvist, I. E. Castelli, R. Christensen, M. Dułak, J. Friis, M. N. Groves, B. Hammer, C. Hargus, E. D. Hermes, P. C. Jennings, P. B. Jensen, J. Kermode, J. R. Kitchin, E. L. Kolsbjerg, J. Kubal, K. Kaasbjerg, S. Lysgaard, J. B. Maronsson, T. Maxson, T. Olsen, L. Pastewka, A. Peterson, C. Rostgaard, J. Schiøtz, O. Schütt, M. Strange, K. S. Thygesen, T. Vegge, L. Vilhelmsen, M. Walter, Z. Zeng, and K. W. Jacobsen, "The atomic simulation environment—a python library for working with atoms", Journal of Physics: Condensed Matter **29**, 273002 (2017).

[63] J. Guénolé, W. G. Nöhring, A. Vaid, F. Houllé, Z. Xie, A. Prakash, and E. Bitzek, "Assessment and optimization of the fast inertial relaxation engine (fire) for energy minimization in atomistic simulations and its implementation in lammps", Computational Materials Science **175**, 109584 (2020).

[64] E. Montahaei, D. Alihosseini, and M. S. Baghshah, "Dgsan: discrete generative self-adversarial network", Neurocomputing **448**, 364–379 (2021).

[65] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: surpassing human-level performance on imagenet classification", in Proceedings of the ieee international conference on computer vision (2015), pp. 1026–1034.

[66] *Pytorch documentation dataparallel package*.

[67] S. Li, Y. Zhao, R. Varma, O. Salpekar, P. Noordhuis, T. Li, A. Paszke, J. Smith, B. Vaughan, P. Damania, and S. Chintala, *Pytorch distributed: experiences on accelerating data parallel training*, 2020.

[68] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: an imperative style, high-performance deep learning library", in *Advances in neural information processing systems 32*, edited by H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (Curran Associates, Inc., 2019), pp. 8024–8035.

[69] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks", arXiv preprint arXiv:1511.06434 (2015).

[70] S. Lee, B. Kim, and J. Kim, "Predicting performance limits of methane gas storage in zeolites with an artificial neural network", Journal of Materials Chemistry A **7**, 2709–2716 (2019).

[71] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, "Grad-cam: visual explanations from deep networks via gradient-based localization", in Proceedings of the ieee international conference on computer vision (2017), pp. 618–626.

[72] X. Chen, Y. Duan, R. Houthooft, J. Schulman, I. Sutskever, and P. Abbeel, "Infogan: interpretable representation learning by information maximizing generative adversarial nets", Advances in neural information processing systems **29** (2016).

[73] A. Sanakoyeu, V. Tschernezki, U. Buchler, and B. Ommer, "Divide and conquer the embedding space for metric learning", in Proceedings of the ieee/cvf conference on computer vision and pattern recognition (2019), pp. 471–480.

[74] S. L. Smith, P.-J. Kindermans, C. Ying, and Q. V. Le, "Don't decay the learning rate, increase the batch size", arXiv preprint arXiv:1711.00489 (2017).

[75] X. Ying, "An overview of overfitting and its solutions", in Journal of physics: conference series, Vol. 1168, 2 (IOP Publishing, 2019), p. 022022.

[76] D. M. Hawkins, "The problem of overfitting", Journal of chemical information and computer sciences **44**, 1–12 (2004).

[77] M. Belkin, D. Hsu, S. Ma, and S. Mandal, “Reconciling modern machine-learning practice and the classical bias–variance trade-off”, Proceedings of the National Academy of Sciences **116**, 15849–15854 (2019).

[78] Y. Yazici, C.-S. Foo, S. Winkler, K.-H. Yap, and V. Chandrasekhar, “Empirical analysis of overfitting and mode drop in gan training”, in 2020 ieee international conference on image processing (icip) (IEEE, 2020), pp. 1651–1655.

[79] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, “Image quality assessment: from error visibility to structural similarity”, IEEE transactions on image processing **13**, 600–612 (2004).

[80] D. Bau, J.-Y. Zhu, J. Wulff, W. Peebles, H. Strobelt, B. Zhou, and A. Torralba, *Seeing what a gan cannot generate*, 2019.